НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

«На правах рукопису»

УДК 004.75: 681.3

До захисту допущено:

Завідувач кафедри

_____ Вадим МУХІН

«___»_____2024 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою

“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”

зі спеціальності 122 "Комп'ютерні науки"

на тему: «Розробка і конфігурація інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирному будинку»

Виконав:
студент ІІ курсу, групи ДА-21мп
Мунтян Даниїл Миколайович _____

Науковий керівник:
доцент, к.т.н.
Кисельов Геннадій Дмитрович _____

Консультант з розділу «Розробка стартап-проекту»:
к.т.н.
Кисельов Генадій Дмитрович _____

Рецензент:
Доцент кафедри АПЕПС к.т.н.,
Світлана Шаповалова _____

Засвічую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2024

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 "Комп'ютерні науки"

Освітньо-професійна програма – "Інтелектуальні сервіс-орієнтовані розподілені обчислювання"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Вадим МУХІН

25 жовтня 2023 р.

ЗАВДАННЯ
на магістерську дисертацію (практика) студенту

Мунтяну Даниїлу Миколайовичу

1. Тема дисертації «Розробка і конфігурація інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирному будинку», науковий керівник дисертації Кисельов Геннадій Дмитрович, к.т.н., доцент кафедри СП, затверджені наказом по університету від __ листопада 2023 р. № ____

2. Термін подання студентом дисертації – 15 грудня 2023 р.

3. Об’єкт дослідження - система контролю споживання електроенергії в багатоквартирному будинку

4. Вихідні дані: мови програмування – Java, фреймворки – Spring,

5. Перелік завдань, які потрібно розробити

5.1. Аналіз і дослідження сучасних систем контролю споживання електроенергії: Це має включати збір інформації про сучасні системи контролю споживання електроенергії, вивчення їх можливостей та обмежень, вивчення технологій, які використовуються в цих системах, та оцінка їх ефективності.

5.2. Розробка проекту інтелектуальної розподіленої системи контролю споживання електроенергії: На основі проведеного аналізу, потрібно скласти технічні вимоги до системи, вибрати оптимальні технології та скласти проект системи. Це включає розробку архітектури системи, визначення основних компонентів та вибір платформи.

5.3. Реалізація і конфігурація системи: На основі затвердженого проекту, потрібно реалізувати систему. Це включає розробку програмного забезпечення, встановлення необхідного обладнання, налаштування системи та її тестування.

5.4. Оцінка ефективності розробленої системи: після тестування системи, потрібно провести її оцінку. Це включає збір даних про споживання електроенергії до та після встановлення системи, аналіз отриманих даних та висновки про ефективність системи.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

Презентація результатів дисертації в Power Point

7. Орієнтовний перелік публікацій: відсутній

8. Дата видачі завдання – 25 жовтня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
	Аналіз і дослідження сучасних систем контролю споживання електроенергії	10 листопада 2023	
	Розробка проекту інтелектуальної розподіленої системи контролю споживання електроенергії	25 листопада 2023	
	Реалізація і конфігурація системи: На основі затвердженого проекту, потрібно реалізувати систему.	30 листопада 2023	
	Оцінка ефективності розробленої системи: після тестування системи, потрібно провести її оцінку.	15 грудня 2023	
	Захист дисертації	19 січня 2024	

Студент Даниїл Мунтян

Науковий керівник Геннадій Кисельов

РЕФЕРАТ

магістерської дисертації Мунтян Даниїла Миколайовича на тему «Розробка і конфігурація інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирному будинку»

Робота виконана на 102 сторінках, містить 53 ілюстрації, 10 таблиць. При підготовці використовувалась література з 24 джерел.

Актуальність дослідження обумовлена необхідністю ефективного управління енергетичними ресурсами у багатоквартирних будинках під час нештатної ситуації, особливо у контексті зростаючих вимог до енергоефективності та надійності постачання електроенергії. Створення інтелектуальної системи контролю, яка забезпечує високу адаптивність та швидку реакцію на енергетичні кризи, відповідає сучасним тенденціям раціонального використання енергії.

Метою роботи є проектування та реалізація розподіленої системи, яка дозволить моніторити та управляти споживанням електроенергії, забезпечуючи безперебійну роботу при аварійних відключеннях та оптимізуючи розподіл енергії між споживачами.

Об'єктом дослідження виступає процес контролю та управління споживанням електроенергії в багатоквартирних будинках з використанням IoT технологій та аналізу потокових даних.

Предметом дослідження є алгоритми та методи оптимізації енергоспоживання, а також принципи розробки розподілених систем та мікросервісів.

Методологія дослідження базується на застосуванні сучасних підходів розробки програмного забезпечення, включаючи гнучкі методики, мікросервісну архітектуру, а також використання технологій для прийняття рішень в реальному часі.

Наукова новизна дослідження полягає у напрацюванні алгоритму, що працює в разі екстреного переключення будинку на резервне живлення та

обробляє потік даних споживання з кожної квартири в реальному часі, мінімізуючи ймовірність аварійного перевантаження резервного генератора, а отже і відключення всього дому від електроенергії, та розробці відповідної інтелектуальної розподіленої програмної частини системи, що допоможе реалізовувати даний алгоритм та відповідає всім вимогам сучасних хмарних рішень.

Перспективи подальшого розвитку дослідження включають розширення функціоналу системи шляхом інтеграції додаткових сервісів для аналізу часових рядів, реалізації інтерфейсу користувача для наочного представлення роботи алгоритму та ручного контролю споживанням кожної квартири, а також адаптації системи для використання в різних середовищах, включаючи хмарні обчислення.

Публікації:

1. Розробка і конфігурація інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирному будинку. / Кисельов Г.Д., Мунтян Д.М.// Системні науки та інформатика: збірник доповідей II Всеукраїнської науково-практичної конференції «Системні науки та інформатика», 4–8 грудня 2023 року, Київ. – К., НН ІПСА КПІ ім. Ігоря Сікорського, 2023. – с. 314-319.

Ключові слова: інтелектуальні системи, енергетична ефективність, IoT, мікросервіси, розподілені системи, енергоменеджмент, автоматизація, ріал-тайм аналіз.

ABSTRACT

Muntian Danyil Mykolaiovych's master's thesis on topic "Development and Configuration of an Intelligent Distributed System for Control of Electricity Consumption in Multi-Apartment Buildings"

The work is completed on 102 pages, contains 53 illustrations and 10 tables, with references to 24 sources

The research's relevance is driven by the need for effective management of energy resources in multi-apartment buildings during emergency situations, especially in the context of growing requirements for energy efficiency and reliability of electricity supply. The development of an intelligent control system capable of adapting quickly and responding effectively to energy crises aligns with contemporary trends in rational energy utilization.

The aim of this work is to design and implement a distributed system capable of monitoring and managing electricity consumption in multi-apartment buildings, ensuring continuous operation during emergency outages, and optimizing energy distribution among consumers.

The object of research is the process of control and management of electricity consumption in multi-apartment buildings using IoT technologies and streaming data analysis.

The subject of research involves algorithms and methods for optimizing energy consumption, and principles for developing distributed systems and microservices.

The research methodology is grounded in modern software development approaches, encompassing agile methodologies, microservice architecture, and technologies for real-time decision making.

The scientific novelty lies in developing an algorithm activated during emergency transitions to backup power, processing real-time consumption data from each apartment. This minimizes the risk of emergency overload of the backup generator, thereby preventing total power outages in the building. It also involves creating an appropriate intelligent distributed software component that conforms to modern cloud solution standards.

Future research prospects include expanding the system's capabilities by integrating additional services for time series analysis, developing a user interface for a clear representation of the algorithm's functionality and manual control over each apartment's consumption, and adapting the system for various environments, including cloud computing.

Publications:

1. Development and configuration of an intelligent distributed system for control of electricity consumption in multi-apartment buildings. / Kyselov G.D., Muntian D.M. // System Sciences and Informatics: Collection of Reports of the II All-Ukrainian Scientific and Practical Conference "System Sciences and Informatics", December 4-8, 2023, Kyiv. – K., ER IPSA KPI, 2023. – pp. 314-319.

Keywords: intelligent systems, energy efficiency, IoT, microservices, distributed systems, energy management, automation, real-time analysis.

ЗМІСТ

ВСТУП

Актуальність розробки та впровадження інноваційних систем контролю споживання електроенергії обумовлені викликами, з якими зіткнулась Україна. Останні події, пов'язані з обстрілами та непередбачуваністю енергопостачання, підкреслили вразливість існуючих систем і підняли питання необхідності впровадження більш гнучких та інтелектуальних рішень для управління енергоресурсами, особливо в багатоквартирних житлових будинках.

У відповідь на ці виклики, дана магістерська робота пропонує концепцію розробки інтелектуальної розподіленої системи, що використовує передові технології інтернету речей для оптимізації споживання електроенергії, забезпечуючи стабільність і надійність електропостачання у критичних ситуаціях.

Проектування даної системи базується на глибокому аналізі сучасного стану енергетичних систем та врахуванні вимог до енергоефективності. Метою системи є не тільки запобігання перевантажень та аварійних відключень джерела резервного живлення, але й розподіл енергоресурсів між споживачами таким чином, щоб кожен користувач отримав максимально можливу кількість енергії, враховуючи його поточні потреби.

Розробка не вимагає радикальних змін в існуючій інфраструктурі, що робить її придатною до впровадження в будь-яких будинках, включаючи старі конструкції. Це є особливо важливим для України, де більшість житлового фонду складають будівлі радянського типу.

Ситуація з недавніми масовими відключеннями спонукала багато громадян та підприємств до придбання генераторів, а це, у свою чергу, визначило потребу в інтелектуальній системі, що здатна в реальному часі регулювати споживання та розподіляти навантаження, мінімізуючи ризики для житлових масивів в умовах обмежених ресурсів.

Передбачається, що реалізація проекту суттєво покращить умови життя в багатоквартирних будинках та стане вагомим внеском у забезпечення енергетичної безпеки країни.

1 ОГЛЯД ІСНУЮЧИХ СИСТЕМ КОНТРОЛЮ СПОЖИВАННЯ ЕЛЕКТРОЕНЕРГІЇ

Schnaider Electric [1] - це визнаний світовий лідер у галузі управління енергією та автоматизації. Їх інноваційні рішення в галузі інтелектуальних будівель забезпечують підвищення ефективності, комфорту та продуктивності в будівлях різних типів. Мета – створення рівних можливостей для максимально ефективного використання енергії та ресурсів, забезпечення прогресу та стійкого розвитку для всіх.

Заснована у Франції, компанія має величезний досвід у створенні інтегрованих рішень у сфері енергетики, що покривають широкий спектр промисловості, від простих перемикачів до складних операційних систем. На рисунку 1.1 зображено логотип компанії.

Рисунок 1.1 - Schneider Electric [1]

ExoStruxure Building – основний продукт у сфері інтелектуальних будівель (Smart House). Це відкрита інноваційна платформа будівель, яка являє собою колаборативне рішення для Інтернету речей (IoT). Ця платформа має архітектуру, що дозволяє робити будівлі всіх типів інтелектуальними. EcoStruxure Building допомагає максимізувати ефективність будівель, оптимізувати комфорт та продуктивність, а також збільшити їхню вартість. Це досягається за рахунок безпечного з'єднання апаратного та програмного забезпечення, а також послуг через Ethernet IP основу.

Schneider Electric також спеціалізується на електрообладнанні. Розширює можливості платформи EcoStruxure за допомогою девайсів наступного покоління, включаючи датчики для житлових приміщень та управління кабелями. Ці інноваційні продукти дозволяють передавати дані від підключеного обладнання, швидше діагностувати та вирішувати проблеми, а також

здійснювати налаштування за допомогою мобільних телефонів, забезпечуючи при цьому високий рівень кібербезпеки.

Ще один великий сервіс, що надає компанія – EcoStruxure Building Advisor, що допомагає покращити загальні витрати власників будівель за рахунок аналітичної звітності та доступу до сервісів на місці або в хмарі. Сервіс безперервно мониторить ефективність будівлі, допомагаючи діагностувати проблеми та надаючи дієві поради для покращення комфорту мешканців та зниження енергетичних і обслуговуючих витрат до 30%.

Siemens Building Technologies [2] - це підрозділ компанії Siemens AG, одного з найбільших світових виробників електроніки та електротехніки. Компанія спеціалізується на розробці інтегрованих рішень для інтелектуальних будівель, пропонуючи повний спектр послуг та технологій для управління будівлями, від HVAC (Heating Ventilation and Air Conditioning) [3] та освітлення до систем безпеки та пожежогасіння.

Siemens пропонує рішення для створення розумних будинків. Ці рішення дозволяють досягти бажаної продуктивності та ефективності будівель різного типу, включаючи шкільні кампуси, лікарні, готелі, аеропорти, музеї, комерційні будівлі та виробничі заводи. Нижче можна побачити ілюстрацію того що пропонує Siemens (рисунок 1.2).

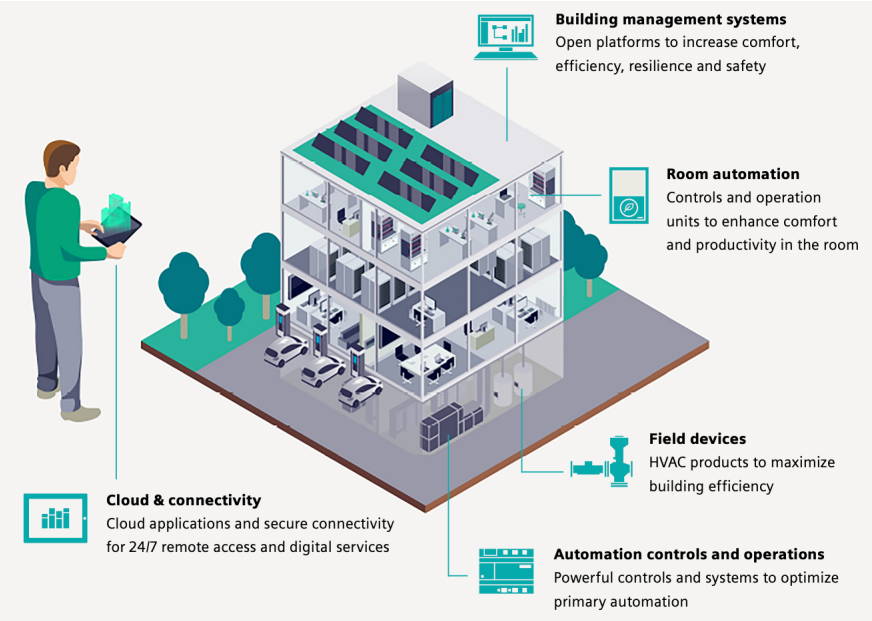

Рисунок 1.2 - Комплексні системи Siemens [2]

Компанія забезпечує неперевершену гнучкість конфігурації завдяки різноманітності користувацьких інтерфейсів, мережевих менеджерів, контролерів та польових пристроїв (“field devices”) [4]. Архітектура дозволяє дистанційно або безпосередньо на місці керувати всіма системами та обладнанням для досягнення оптимальної ефективності розумного будинку.

Будинки відповідають за близько 36% глобального споживання енергії та майже 40% глобальних викидів парникових газів протягом їх життєвого циклу. Siemens підкреслює, що навіть невеликі покращення в операціях та споживанні енергії мають значення для просування декарбонізації. Компанія розробляє рішення, які сприяють енергоефективності як у будівництві з нуля, так і в існуючих будинках.

Iotji - це інноваційна українська компанія [5], яка спеціалізується на розробці та імплементації рішень в сфері Інтернету речей (IoT) для управління енергоспоживанням. Їх місія, подібно до інших, полягає у створенні інтелектуальних систем, які дозволяють оптимізувати використання ресурсів, забезпечуючи економію та підвищення ефективності енергетичних систем. Логотип компанії можна побачити на рисунку 1.3.

Рисунок 1.3 - Українська компанія Iotji [5]

Iotji пропонує досить широкий спектр продуктів та послуг. Це включає в себе розумні лічильники, датчики споживання, системи моніторингу та управління навантаженням, що використовують сучасні технології, такі як LoRaWAN (Long Range Wide Area Network) [6], для надійної та безпечної передачі даних. Тобто рішення, що пропонує Iotji дозволяють встановити якісний моніторинг виробничих потужностей, автоматизувати процеси та істотно підвищити прибутковість бізнесу.

Аналіз та порівняння. Отже, підсумовуючи, можна сказати, що Siemens та Schneider Eelectric орієнтуються на великі бізнеси та будівельні компанії, які інтегрують «розумні» рішення на етапі проектування або будівництва. Iotji, в свою чергу, звертається до середнього та великого бізнесу в Україні, пропонуючи рішення для вже існуючих об’єктів і, що важливо, без необхідності їхньої попередньої модернізації.

Такі рішення як Siemens Building Technologies та ExoStruxure Building пропонують масштабні системи, здебільшого для Smart House, та мають вимоги до певних стандартів та характеристик будівлі. Рішення від Iotji більш гнучкі та адаптивні, оскільки вони можуть бути застосовані до вже існуючих систем.

Система що представлена в даній роботі розроблена для впровадження в будь-який багатоквартирний будинок, налаштований на роботу з резервним генератором. У випадку відключення енергії, система автоматично працює на запобігання перенавантаженню генератора, забезпечуючи хоч і обмежене, але стабільне електропостачання, мінімізуючи ймовірність його повного відключення.

Отже, в ході аналізу не було знайдено прямих аналогів системи, що представлена в даній роботі. Більшість існуючих рішень зосереджені на моніторингу, енергоефективності та зниженні затрат для бізнесу, тоді як дана робота орієнтована на ті комунальні підприємства або управляючі компанії, що прагнуть мати надійний інструмент для запобігання аварійним ситуаціям, що можуть призвести до збоїв у резервному енергопостачанні багатоквартирних будинків.

2 ТЕОРЕТИЧНА ОСНОВА ПРОЕКТУ

2.1 Постановка задачі та вхідні дані

В контексті даного дослідження розглядається система з використанням стандартного промислового генератора електроенергії, інтегрованого з багатоповерховим будинком та налаштованого для автоматичного включення у випадку екстрених ситуацій, зокрема при відключенні від централізованої електромережі.

Маємо мережу IoT пристроїв «Smart Box» встановлених в кожній квартирі та з'єднаних з IoT Gateway – невеликим локальним сервером, розташованим в будинку, який агрегує дані та має вихід до мережі Інтернет. Детальніше про локальну конфігурацію будинку викладено в розділі «Технічна складова системи». На даному етапі важливо знати, що Smart Box – це звичайний «польовий» девайс, що вміє робити наступні речі.

1. Вимірювати потужність споживання та надсилати ці дані на локальний IoT Gateway.
2. Відключати та підключати живлення квартири по команді від IoT Gateway.
3. Передача на IoT Gateway сигналу «готовності» від мешканців квартир, який ініціюється за допомогою спеціальної кнопки після здійснення заходів щодо зниження споживання електроенергії.

Додатково в систему інтегрований IoT-пристрій на вузлу підключення генератора до будинку, який відправляє сигнали про різні статуси системи:

1. BLACKOUT - маємо відключення від міської мережі.
2. GENERATOR_ACTIVATED - після того як спрацює система автоматичного вводу резерву (АВР) генератора [посилання!], він буде запущений.
3. OVERLOADED - генератор перевантажено та відбулося аварійне відключення (буде автоматично запущений знову найближчим часом).
4. NORMALIZED – відбулось відновлення міського живлення та система АВР генератора перемкнулась на загальну систему електропостачання.

Важливим елементом системи є ввідний автомат кожної квартири [7] (рис. 2.1), який визначає максимальну можливу потужність споживання, фактично

встановлюючи номінальне навантаження для кожної індивідуальної квартири P_{user_i}, де i – номер квартири. Тоді загальне номінальне навантаження усіх квартир можна виразити як:

$$P_{user} = \sum_{i=1}^{n} P_{user_i} \quad (1)$$

Де n – кількість квартир в будинку.

Рисунок 2.1 - Приклад ввідного автомату квартири (номінально потужність 7 кВт)

Далі вводимо позначення для номінальної потужності генератора як P_{gen} і визначаємо максимальну потужність, яка потрібна для задоволення загальнодомових потреб (освітлення, паркінг, шлагбаум, котельня тощо) як P_{com}.

Враховуючи, що максимальна потужність, яку можна використовувати з генератора, повинна бути меншою за його номінальну потужність, ми вводимо коефіцієнт k_{gen}, який за замовчування дорівнює 0.8 (80% від номінальної потужності генератора). Цей коефіцієнт дозволяє налаштовувати допустиме навантаження на генератор, забезпечуючи його ефективну та безпечну роботу.

Алгоритм, що представлено у розділі «Опис алгоритму роботи», включає в себе логіку підключення відключених квартир до системи живлення від генератора. Підключення квартири №i до мережі дозволяється тільки тоді, коли

це не призведе до перевищення максимально допустимого навантаження генератора.

$$P_{gen}^{current} + P_{user_i} \leq k_{gen} * P_{gen} \quad (2)$$

Де $P_{gen}^{current}$ – поточне навантаження генератора. Тобто використовується песимістичний підхід - завжди припускається найгірше – те, що відключена квартира потенційно почне споживати максимум.

Встановлюємо X – коефіцієнт ліміту потужності для кожної квартири при живленні від резервного джерела електроенергії. Він визначає максимально прийнятний рівень споживання електроенергії для кожної квартири, який є часткою її номінальної потужності. Цей коефіцієнт використовується для створення прогнозованого та безпечного режиму роботи системи. Квартири, що споживають енергію понад встановлений цим коефіцієнтом ліміт, класифікуються алгоритмом як порушники встановлених правил споживання.

Отже, враховуючи вищезазначене, можна виписати нерівність, що повинна виконуватися завжди.

$$k_{gen} * P_{gen} \geq X * \left(\sum_1^n P_{user_i} - \max_i P_i\right) + \max_i P_i + P_{com} \quad (3)$$

Де $\max_i P_i$ – максимальне значення номінального споживання серед квартир.

Так як ми завжди зацікавлені в тому, щоб P_{gen} був менше (відповідно генератор буде дешевше), а значення X хочемо тримати якомога найбільшим (відповідно кожній квартирі ліміт буде більший), то урівняючи ліву і праву частину, отримаємо формулу розрахунку коефіцієнту ліміту потужності для кожної квартири при екстреній ситуації та, відповідно, живленні від генератора.

$$X = \frac{k_{gen} * P_{gen} - \max_i P_i - P_{com}}{\sum_1^n P_{user_i} - \max_i P_i} \quad (4)$$

На практиці, це означає, що перед початком відключення електроенергії від основної мережі, система визначає максимально можливу потужність споживання для кожної квартири на базі встановленого коефіцієнту X. Це

допомагає підтримувати стабільну роботу генератора та запобігати його перевантаженню.

Наприклад, маємо будинок з n = 20 квартирами. П’ять з них мають номінальну потужність 3500 Вт, п’ять наступних – 5000 Вт, інші п’ять – 6500 Вт, останні п’ять – по 8000 Вт. Таким чином, $\max_i P_i = 8000$ Вт та $\sum_1^n P_{user_i} = 5 * 3500 + 5 * 5000 + 5 * 6500 + 5 * 8000 = 115000$ Вт.

Нехай маємо дизельний генератор потужністю $P_{gen} = 50$ кВт і бажану робочу потужність генератора не більше ніж 80% від номінальної, тобто $k_{gen} = 0.8$, відповідно допустиме навантаження на генератор – 40 кВт.

Припустимо, що на всі загальнодомові потреби потрібно $P_{com} = 1000$ Вт.

Тоді можемо розрахувати коефіцієнт ліміту потужності кожної квартири:

$$X = \frac{k_{gen} * P_{gen} - \max_i P_i - P_{com}}{\sum_1^n P_{user_i} - \max_i P_i} = \frac{0.8 * 50000 - 8000 - 1000}{115000 - 8000} = 0.28$$

Це означає, що кожна квартира може споживати максимум 28% від своєї номінальної потужності. Наприклад, для квартири з номінальною потужністю 3500 Вт, максимальне дозволене споживання становитиме 980 Вт (3500 Вт $*$ 0.28), а для квартири з потужністю 8000 Вт – це буде 2240 Вт.

Очевидно, що якщо номінал генератора досить великий, то в системі не буде багато сенсу, бо і обмежень на споживання не буде. Алгоритм потрібен саме для тої ситуації, коли номінал генератора набагато менший за всі потреби дому, як це часто буває на практиці.

2.2 Технічна складова системи

2.2.1 IoT технології та їх застосування

Що це таке та чому це важливо. Інтернет речей (IoT) [4] представляє собою мережу фізичних об'єктів, оснащених датчиками, програмним забезпеченням та різними технологічними компонентами. Ці «речі» здатні взаємодіяти одна з одною, збираючи та обмінюючись даними, які потім можуть бути передані на інші пристрої в системі або інші мережі через Інтернет. В основі

IoT лежить ідея про те, що фізичні предмети можуть підключатися до Інтернету для обміну інформацією чи отримання команд.

Концепція Інтернету речей існує вже тривалий час, проте лише нещодавні дослідження та розробки зробили її реально практичним. Якщо близько 15 років тому використання машинного спілкування обмежувалося переважно великими промисловими комплексами, то зараз IoT-пристрої стали частиною повсякденного життя. Різноманітність підключених пристроїв до Інтернету надзвичайно широка, від простих побутових гаджетів до складних промислових систем та екологічних станцій.

Прогрес в області обчислювальних можливостей, хмарних технологій, аналітичних інструментів та бездротових засобів зв'язку сприяв тому, що фізичні об'єкти здатні обмінюватися даними з мінімальним втручанням з боку людини. Сучасні цифрові системи забезпечують реєстрацію, моніторинг та керування взаємодіями між пристроями, вирішуючи при цьому складні завдання і сприяючи ефективному взаємодії цифрового та фізичного світів.

Розвиток Інтернету речей відкрив шлях до більш доступних і енергоефективних технологій. Зокрема, впровадження доступних та високонадійних датчиків зробило IoT ключовим рішенням для широкого кола виробників та бізнесу. Як уже було зазначено в цій дисертації, розвиток IoT значно розширив можливості застосування інтелектуальних розподілених систем, відкриваючи нові горизонти для вирішення практичних завдань. На рисунку 2.1 представлена типова архітектура системи, заснованої на IoT у контексті Smart House.

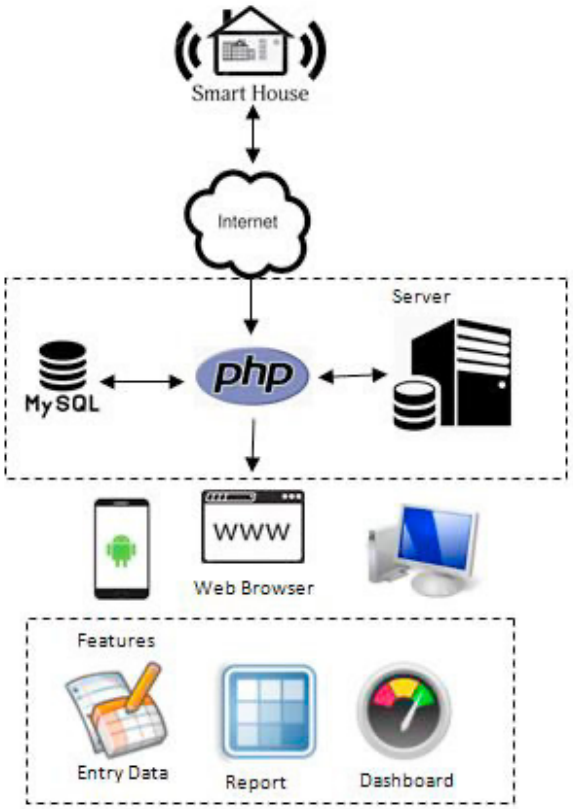

Рисунок 2.1 - Базовий дизайн системи IoT [4]

Інтернет речей використовує широкий спектр мережевих протоколів для забезпечення доступу до Інтернету. Ця різноманітність спрощує інтеграцію різних типів датчиків у хмарні системи та з іншими пристроями, що сприяє ефективному обміну даними. Поява доступних хмарних платформ значно полегшила процес отримання користувачами та підприємствами доступу до потрібної інфраструктури для масштабування своїх систем без необхідності безпосереднього управління ними.

Зв'язок. Дана робота фокусується на розробці програмного забезпечення, проте технічний аспект також відіграє важливу роль. Як уже згадувалося, пристрої Smart Box формують локальну мережу, яка спілкується з IoT Gateway. В цілому, екосистема Інтернету речей складається з різноманітних пристроїв, що не впливають один на одного та функціонують з метою збору та передачі широкого діапазону даних. Ключовим елементом для їх ефективної роботи є уніфікований протокол передачі даних. Серед найбільш відомих IoT протоколів можна виділити такі як Bluetooth, Wifi, MQTT, Zigbee, , NFC, DDS, LTE і LoRaWAN. Останній з них є особливо важливим в контексті нашого

дослідження, адже саме LoRaWAN оптимально підходить для забезпечення зв'язку між IoT Gateway та Smart Box’ами в будинку.

LoRaWAN є одним із провідних стандартів у сфері LPWAN (Low Power Wide Area Network) [6] - мереж з широким радіусом дії та низьким споживанням енергії. Цей стандарт отримав популярність завдяки зусиллям міжнародної некомерційної асоціації LoRa Alliance, ключовими учасниками якої є такі компанії, як Semtech (NASDAQ:SMTC), IBM та Cisco, які відіграли значну роль у розробці та просуванні протоколу LoRaWAN.

Особливістю стандарту LoRaWAN є його відкритість, що створює конкурентне середовище серед постачальників обладнання та забезпечує високий рівень сумісності між пристроями від різних виробників. Ця відкритість гарантує, що користувачі не стануть заручниками пропрієтарних технологій, оскільки вони мають свободу вибору рішень від різних постачальників.

LoRaWAN, або Long Range Wide Area Network, призначений для ефективної передачі даних на далекі відстані і спрямований на підтримку великої кількості низькопотужних пристроїв, які використовуються в промисловості. Цей стандарт оптимально підходить для створення широкої мережі пристроїв, що вимагають мінімального споживання енергії, і забезпечує надійний зв'язок навіть на великих відстанях.

Відомі формати бездротового зв'язку, такі як GSM, 3G, LTE, та WiFi, зазвичай націлені на забезпечення високошвидкісної передачі даних для активних користувачів і використовують ліцензовані радіочастоти для забезпечення стабільної комунікації на порівняно невеликі відстані. LoRaWAN навпаки, розроблений із фокусом на машини та датчики, які функціонують у неліцензованому спектрі, що усуває необхідність урегулювання з органами влади та знижує вимоги до пропускної спроможності каналу передачі даних.

Завдяки своїм характеристикам, стандарт LoRaWAN стає ідеальним вибором для малопотужних датчиків і пристроїв, забезпечуючи передачу даних на відстані до 10 км у відкритій місцевості та 1-2 км у міських умовах, а також проникнення сигналу в важкодоступні місця, як-от колодязі чи підвали.

Застосування індивідуально налаштованого коефіцієнта розширення спектра (Spreading Factor) дозволяє регулювати швидкість і дальність передачі даних, забезпечуючи оптимальний баланс між швидкістю передачі, дальністю та стійкістю до радіоперешкод.

На рисунку 2.2 ілюструється структура зразкової мережі LoRaWAN.

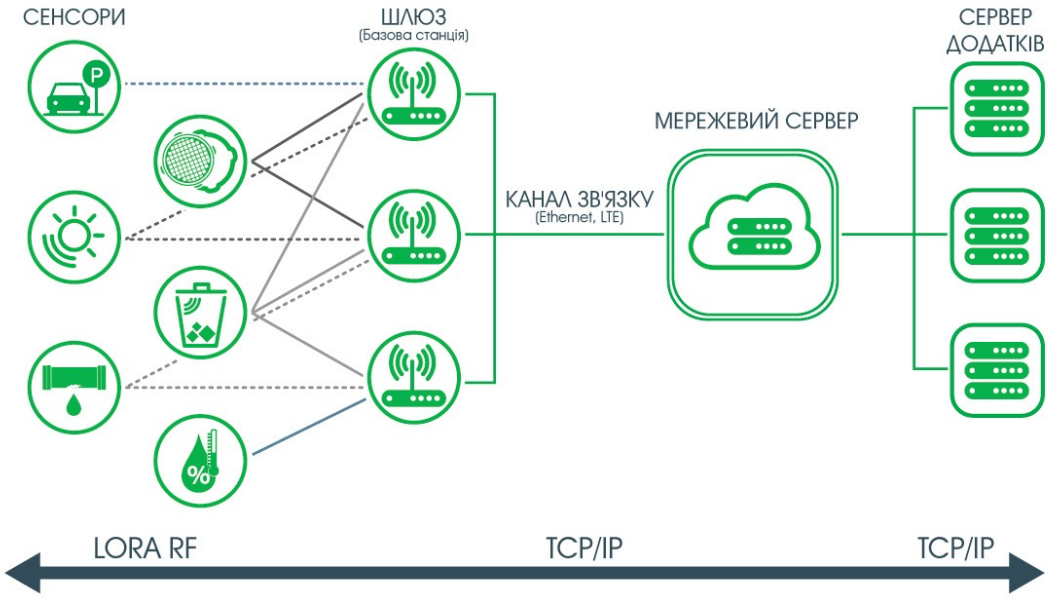

Рисунок 2.2 - Мережа LoRaWAN [4]

У рамках мережі LoRaWAN, кінцеві вузли, що є радіомодулями, відправляють інформацію до центрального вузла (шлюзу або базової станції). Далі ці дані передаються через високошвидкісний комунікаційний канал до мережевого сервера, який розподіляє отримані дані між серверами різних застосунків. У контексті нашого проекту використовується єдиний шлюз — IoT Gateway.

Застосування IoT охоплює широкий спектр галузей та сценаріїв використання, від промисловості до побутових потреб. Основним принципом є збір даних від безлічі датчиків, їх обробка та аналіз, що дозволяє автоматизувати процеси, підвищити ефективність використання ресурсів та оптимізувати управління системами.

У промисловості IoT сприяє створенню розумних виробничих ліній, де машини та обладнання можуть спілкуватися між собою, прогнозувати несправності та оптимізувати виробничі процеси. У сільському господарстві IoT дозволяє вести точне землеробство, контролюючи стан посівів та вологість грунту, що призводить до збільшення врожайності та зниження витрат. У міському управлінні IoT втілює концепцію «розумних міст», де різні датчики допомагають регулювати дорожній рух, керувати вуличним освітленням та оптимізувати використання енергетичних ресурсів.

У сфері охорони здоров'я IoT набуває особливого значення, надаючи можливість відстежувати стан пацієнтів в реальному часі, забезпечувати дистанційне моніторинг та управління медичними приладами. У побуті IoT пропонує зручність та економію завдяки розумним термостатам, освітлювальним системам та енергоефективним приладам, які можуть автоматично адаптуватися до потреб користувачів.

Інноваційність IoT полягає не лише у використанні новітніх технологій, але й у способі зміни підходів до управління ресурсами та сервісами. Відкритість, гнучкість і можливість масштабування — основні переваги IoT, які забезпечують його широке впровадження у сучасному світі. Ці характеристики, в поєднанні з широкою доступністю даних та здатністю до їх аналітичної обробки, відкривають нові можливості для інновацій у всіх сферах діяльності.

2.2.2 АВР система генератора

Автоматичний ввід резерву (АВР) [8] – пристрій, призначений для автоматичного перемикання споживача на резервне джерело електропостачання у разі відключення основного. Процес автоматичного ввімкнення резерву називається «робота АВР» [8]. На рисунку нижче представлено приклад схеми АВР.

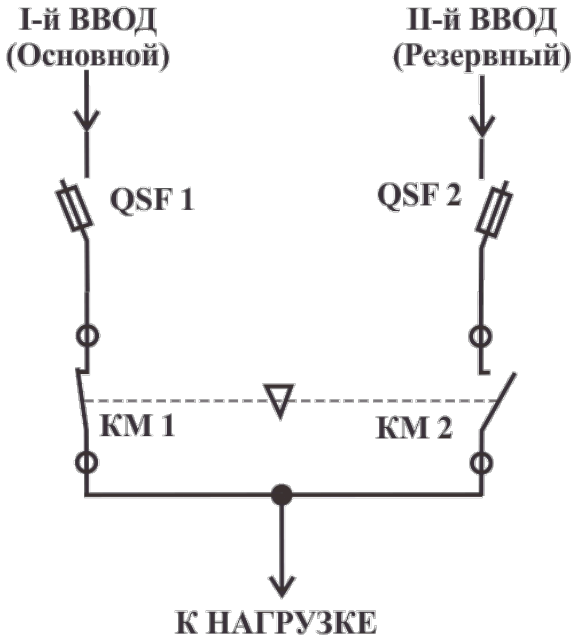

Рисунок 2.3 - Схема АВР [8]

Для запуску роботи електрогенераторів існують різноманітні методики: ручний старт, використання електростартера, дистанційне управління за допомогою пульта або застосування автоматики через систему АВР (автоматичне введення резерву).

Традиційний підхід передбачає ручний запуск генератора, що може вимагати певних фізичних зусиль та залежить від зовнішніх умов, таких як температура повітря. Потрібно враховувати, що для більш потужних агрегатів необхідно докласти більше зусиль для запуску. Варто також пам'ятати, що в період між відключенням електроенергії та запуском генератора усі електроприлади в приміщенні залишаються без живлення.

Альтернативою є використання електростартера, який значно спрощує процедуру запуску – достатньо обернути ключ для ініціації роботи генератора, але, як і в попередньому випадку, до моменту запуску усі прилади залишаються вимкненими.

Більш сучасний та зручний метод – це дистанційне управління за допомогою пульта, що дозволяє активувати генератор натисканням кнопки, не виходячи з дому, що істотно зменшує час, протягом якого електроприлади залишаються без живлення.

Втім, незважаючи на переваги вищезазначених методів, вони все одно вимагають присутності людини для запуску генератора. Це створює певні складнощі у випадках, коли необхідно швидке відновлення електропостачання, але немає можливості фізично вплинути на систему, наприклад, коли власник відсутній вдома.

Найбільш ефективним та передовим методом, який гарантує неперервну роботу домашніх та офісних електроприладів без участі людини, є інтеграція системи автоматичного введення резерву (АВР) з електрогенератором.

Система автоматичного введення резерву (АВР) представляє собою спеціалізований пристрій, що розрахований на роботу з двома джерелами електроенергії. Головний ввід пристрою є постійно активним для приєднання навантажень, тоді як резервний ввід активується виключно у випадках, коли основне джерело виходить з ладу або стає недоступним. АВР автоматично визначає пріоритетність джерела енергії на основі наявності стандартних параметрів мережі, управляючи переключенням з необхідними затримками для безпеки процесу. Зазвичай перемикання виконується з мінімальною затримкою, щоб перерви в електропостачанні були найкоротшими. Число альтернативних джерел електропостачання може бути розширено для підвищення рівня резервування.

АВР є комплексним електротехнічним пристроєм, який включає в себе один або декілька перемикаючих комутаційних пристроїв (ПКО) для забезпечення можливості перемикання між різними джерелами електроживлення. Критерії та характеристики для цих комутаційних пристроїв регламентовані відповідними стандартами, такими як американський UL1008, європейський IEC 60947-6-1:2005 та український ДСТУ IEC 60947-6-1:2007, який є точним аналогом європейського стандарту [9]. Комутаційне обладнання працює таким чином, що дозволяє від'єднувати навантаження від одного джерела електроенергії та підключати їх до іншого, забезпечуючи таким чином безперервність електропостачання.

Комутаційні пристрої класифікуються за способом управління та функціональними можливостями:

- Ручні комутаційні пристрої (РКП) – це такі, що керуються оператором безпосередньо, шляхом механічної дії.
- Дистанційно керовані комутаційні пристрої (ДКП) – це комутаційні пристрої, які можна активувати з віддаленого місця.
- Автоматичні комутаційні пристрої (АКП) – це комутаційні пристрої, що виконують свої функції без необхідності втручання людини.
- Модифіковані комутаційні пристрої (МКП) – це такі, що включають елементи, які відповідають іншим нормативам серії IEC 60947 і спеціалізовані під конкретні завдання.

Перемикання може бути реалізовано з використанням затримки чи без неї, і може включати позицію "відключено". Це критично важливо для функціонування запропонованої системи. Наприклад, генератор може бути задіяний після 2 хвилин після відключення мережі, що дозволяє пересвідчитись у відсутності мережевого живлення і забезпечити мешканцям час для відключення непотрібних електроприладів, відповідно до вимог системи щодо обмеження споживання.

Комутаційні пристрої також різняться за здатністю опиратися короткому замиканню:

- Клас PC: такі комутаційні пристрої здатні включати та витримувати струми короткого замикання, але не призначені для їх вимикання.
- Клас CB: комутаційні пристрої оснащені пристроєм для розчіплення максимального струму і їх основні контакти спроможні активувати та вимикати струм короткого замикання.
- Клас CC: комутаційні пристрої можуть включати і витримувати струми короткого замикання, але не призначені для їх вимикання. Такі пристрої базуються на апаратах, що відповідають вимогам стандарту IEC 60947-4-1 [9].

Конструктивні характеристики робочого механізму комутаційного обладнання передбачають наявність таких елементів:

- Механізм блокування, який забезпечує неможливість одночасного з'єднання з основним та альтернативним джерелами електроенергії, незалежно від зовнішніх обставин. Цей механізм має функціонувати безвідмовно навіть у разі демонтажу зовнішніх дверцят чи інших доступних частин.
- Для комутаційних пристроїв класу PC/CC робочий механізм має гарантувати, що електричне навантаження не буде виключене на тривалий час з обох джерел електропостачання. Проте, конструкція може передбачати короткочасне відключення під час перемикання, що дозволяє завершити процес без ризику для системи. В окремих випадках може бути забезпечене вихідне положення механізму перед початком його роботи.
- Комутаційні пристрої класу CB можуть мати замірений період відключення та/або спеціальне положення для відключення.
- У випадку, коли дія головних контактів забезпечується електромеханічним пристроєм, замикання та розмикання цих контактів має відбуватися плавно, без раптових зупинок, що мінімізує ризик негативного впливу на електромережу.

Автоматика управління АВР є невід'ємною частиною системи, яка забезпечує її автономне функціонування. Ключовим елементом в цьому процесі є електронний блок керування, який несе відповідальність за активацію комутаційного механізму. Важливо обрати надійну автоматику, оскільки саме вона часто стає вразливим місцем у системі АВР. Блок керування може бути інтегрованим безпосередньо у комутаційний механізм або ж розташованим окремо, пов'язаним з основним вузлом сполучним кабелем, що є більш поширеним рішенням. Подача енергії до блоку управління має здійснюватися одночасно від основного та резервного джерел, а в разі зникнення напруги - від вбудованого акумулятора або батареї, щоб забезпечити збереження налаштувань системи.

Для регулювання параметрів роботи АВР використовується спеціальна програмна настройка. Ключові параметри, такі як напруга та частота електропостачання по кожній фазі, встановлюються з певними граничними величинами. Наприклад, для напруги у 380 В може бути встановлений допуск від -15% до +10% для кожної фази окремо.

Значущість мають часові затримки, які регулюють процес перемикання АВР між основним і резервним джерелами енергії. Не завжди потрібно відразу перемикати на резервне джерело при короткочасному зниженні напруги - бажано надати системі можливість для відновлення основного живлення, щоб уникнути подвійного перехідного процесу. Блок управління відповідає за встановлення параметрів джерел електроенергії та регулювання часових налаштувань роботи АВР.

Отже, навіть за наявності незалежних джерел живлення, таких як ДБЖ чи дизельні генератори, безперебійність електропостачання залежить безпосередньо від надійності АВР. Неякісна робота цього пристрою може призвести до критичних ситуацій для споживачів.

2.2.3 Схема технічної інфраструктури

На рисунку нижче проілюстровано технічну реалізацію системи.

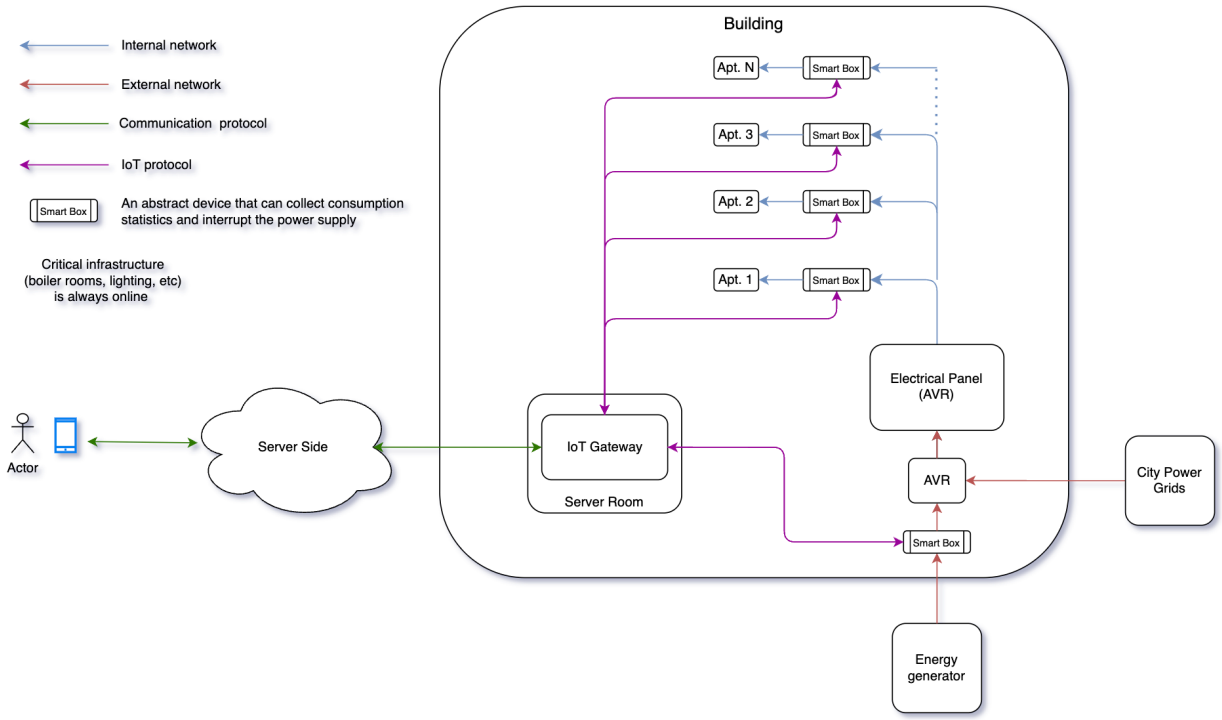

Рисунок 2.4 - Hardware setup

Розберемо всі сутності, представлені вище.

Smart Box – це ключовий елемент системи, інноваційний IoT пристрій, що надає змогу вимірювати споживання електроенергії у кожній квартирі та у разі необхідності переривати електропостачання. Кожен Smart Box має унікальний ідентифікатор, який відповідає номеру квартири, та налаштований на роботу з однофазним, двофазним або трифазним типом електропостачання, забезпечуючи гнучкість та точність у відповідності до специфікацій споживання. Слід зазначити, що при відключенні електроенергії, Smart Box отримує живлення від внутрішнього акумулятору, до того моменту поки не запуститься генератор або не відновиться основне джерело електроенергії.

Ядро системи становить мережа цих пристроїв Smart Box (LoRaWAN), яка забезпечує зв'язок між індивідуальними квартирами та централізованим

сервером IoT Gateway – програмним сервісом, що запущено на одноплатному комп’ютері Raspberry Pie [10], який облаштовано в коморі будинку та оснащений джерелом безперебійного живлення та неперервним доступом до інтернету. На рисунку 2.5 можна побачити його апаратну частину.

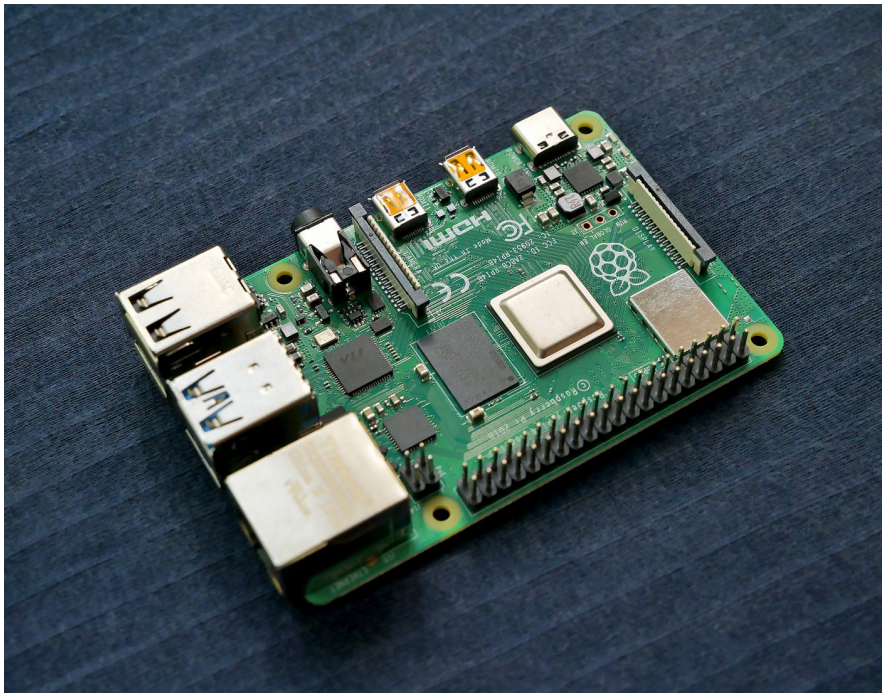

Рисунок 2.5 - Одноплатний комп’ютер Raspberry Pi [10]

На практиці, IoT Gateway – це назва програмного мікросервісу, що написано на Java (Spring) і запущено на Raspberry Pi, який виступає агрегатором всіх даних з датчиків та входом в систему технічного обладнання будинку.

Його основні задачі:

- Отримання IoT даних в домі.
- Агрегація, підготовка і відправка даних Smart Box до хмарного застосунку по мережі Інтернет (в режимі real-time, тобто частота ~1 раз на секунду).
- Сигналізування до програми про наступні події: відключення світла, запуск генератора, аварійне відключення генератора, та відновлення електроенергії від основного джерела.

- Отримання команд від хмарного застосунку, як наприклад підключення або відключення квартири від мережі.
- Відправка отриманих команд до Smart Box'ів відповідних квартир.

Таким чином, IoT Gateway є точкою входу як від програмної частини до апаратної, так і навпаки. Приклад даних, що Gateway регулярно надсилає до програмного застосунку зображено на рисунку 2.6.

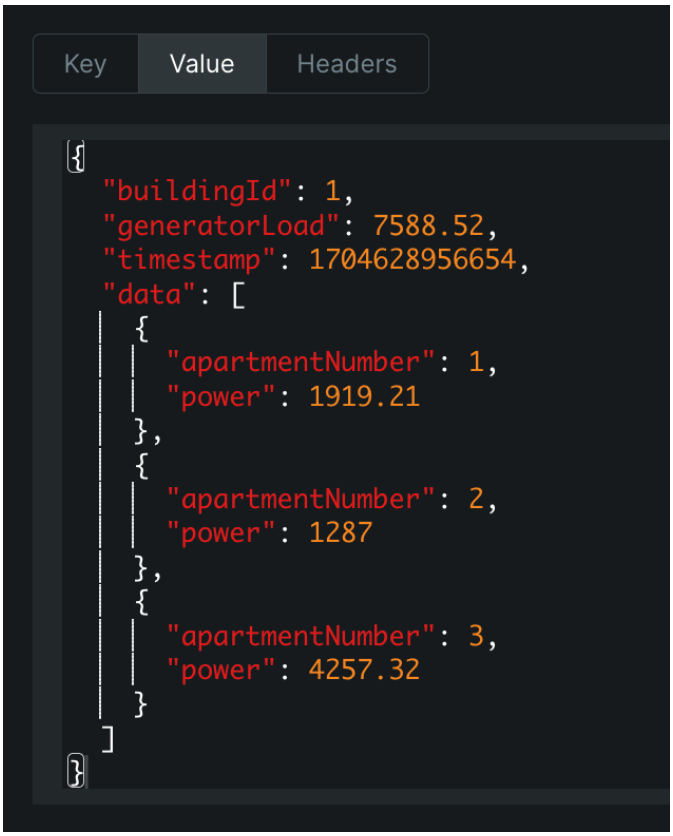

Рисунок 2.6 - Приклад Smart Box даних для трьох квартир

Критично важливі системи, такі як котельні та освітлення, постійно отримують живлення для забезпечення стабільності життєво необхідних функцій.

Користувачі системи, або актори, здебільшого мешканці будинку, отримують інтерактивні сповіщення та застереження від системи. Використовуючи ці інструменти, мешканці можуть реагувати на повідомлення про високе споживання в реальному часі, що дозволяє їм приймати необхідні

заходи для зниження споживання або підготовки до можливого відключення електроенергії.

Енергогенератор – це джерело генерації електроенергії для будинку [11]. На даному етапі ми розглядаємо звичайний промисловий генератор, але система відкрита до інтеграції з різними джерелами енергії. Приклад такого промислового генератора можна побачити на рисунку 2.7.

Рисунок 2.7 - Промисловий дизельний генератор PRAMAST POWER 100 кВт [11]

Зазначена технічна архітектура вирізняється своєю масштабованістю та легкістю інтеграції в існуючі енергетичні системи, що робить її ідеальною не лише для нових Smart house, але й для модернізації старих будинків. Її універсальність полягає в незалежності від віку чи типу будівлі, надаючи можливість кожному житловому комплексу стати більш енергоефективним та відповідальним у використанні енергоресурсів.

Ця система є прикладом того, як передові технології можуть сприяти створенню більш сталого та енергетично збалансованого суспільства, відповідаючи на виклики енергетичної кризи та високого споживання ресурсів, і в той же час забезпечуючи високий рівень комфорту для своїх користувачів.

2.3 Програмна складова системи

2.3.1 Функціональні та нефункціональні вимоги

Систематичне розуміння та формулювання вимог до програмної складової системи становить фундамент, на якому будується вся архітектура будь-якої інтелектуальної розподіленої системи. Визначення вимог є одним із найбільш критичних етапів в процесі розробки, адже від точності та повноти їх опису залежить кінцева функціональність продукту, його якість та вартість реалізації. Ці вимоги відіграють вирішальну роль у спілкуванні між замовниками, розробниками та іншими зацікавленими сторонами, забезпечуючи чітке бачення кінцевої мети проекту.

Тому функціональні та нефункціональні вимоги [12] займають стратегічно важливе місце в загальному пайплайні розробки, оскільки забезпечує підґрунтя для всіх подальших етапів: від проектування та кодування до тестування та підтримки системи. Вимоги слугують як орієнтир для встановлення пріоритетів, визначення обсягу проекту та управління змінами. Вони дозволяють виявити та уникнути потенційних непорозумінь або недоліків, які можуть з'явитися на пізніших стадіях реалізації.

Кожен проект починається з ідентифікації та аналізу цих двох категорій вимог, що дозволяє створити повноцінний проектний документ, який стане відправною точкою для всіх членів команди. Отже, в цьому розділі ми детально розглянемо вимоги, які дозволять нам гарантувати, що розроблена система буде не тільки виконувати встановлені завдання, а й задовольняти строгі стандарти якості. Не менш важливим є розуміння різниці між функціональними та нефункціональними вимогами. На рисунку 2.8 проілюстровано досить простий та наочний приклад таких вимог.

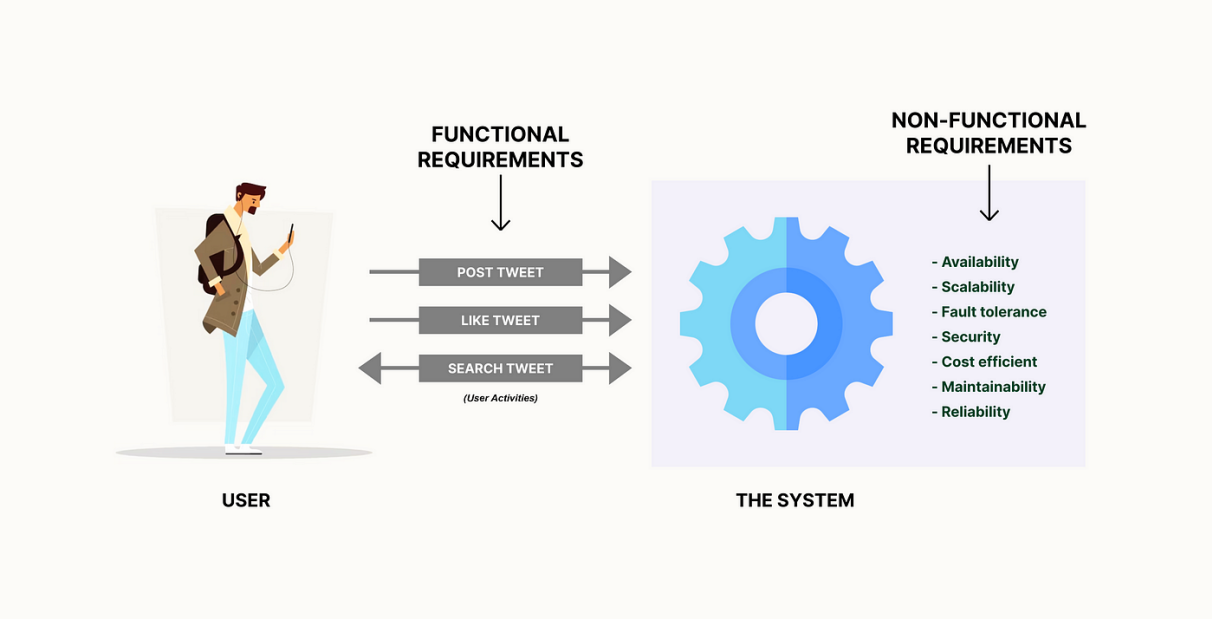

Рисунок 2.8 - Приклад вимог до програмного забезпечення [12]

Функціональні вимоги, які інколи позначаються абревіатурою FR, є критичними параметрами, що визначають основні операції і процедури, які програмна система повинна виконувати. Вони детально описують задачі та процеси, які система має реалізувати, і є основою для взаємодії між програмою та її користувачами. Основні характеристики функціональних вимог включають:

- Специфічність: вимоги чітко формулюються для уникнення будь-яких двозначностей, описуючи взаємодії та результати дій системи.
- Можливість перевірки: можливість підтвердження виконання вимог через тестування, що забезпечує відповідність програми очікуванням.
- Орієнтованість на користувача: вимоги відображають потреби кінцевих користувачів, що забезпечує доцільність і корисність програмного продукту.
- Адаптивність: гнучкість для змін відповідно до розвитку потреб проекту та зворотного зв'язку від користувачів.

Нефункціональні вимоги, відомі також як NFR, доповнюють функціональні вимоги, визначаючи обмеження та атрибути якості, з якими система має функціонувати. Ці вимоги описують "як" система виконує задані функції, включаючи аспекти продуктивності, безпеки та зручності. Основні характеристики нефункціональних вимог включають:

- Якісний орієнтир: акцент на атрибутах якості, таких як надійність, продуктивність та безпека системи.
- Всеосяжність: застосовуються до всієї системи, формуючи її загальну архітектуру та поведінку.
- Стабільність: відзначаються стійкістю в часі і менш схильні до змін, ніж функціональні вимоги.
- Вимірюваність: хоча можуть бути складнішими для кількісного визначення, їх все ж таки можливо оцінити та протестувати.

Зібрання і аналіз як функціональних, так і нефункціональних вимог є невід'ємною частиною процесу розробки програмного забезпечення. Вони визначають рамки проекту і слугують основою для всіх подальших етапів розробки — від проектування архітектури до тестування та впровадження системи. Ці вимоги не тільки впливають на функціональність продукту, але й на його здатність задовольняти потреби користувачів, витримувати навантаження та забезпечувати безпеку даних.

Функціональні вимоги в основному спрямовані на те, що користувачі бачать і з чим взаємодіють безпосередньо, тоді як нефункціональні вимоги забезпечують надійність, ефективність та готовність системи до масштабування. Ретельне визначення та документування обох типів вимог є критичним для успішного запуску та довгострокового утримання програмного продукту на ринку [12].

Таким чином, докладне врахування функціональних та нефункціональних вимог є фундаментальною передумовою для створення високоякісного та конкурентного програмного забезпечення. Вони служать як керівництво для команди розробників, встановлюючи чіткі цілі та стандарти, якими повинна керуватися команда під час всього процесу розробки, і врешті-решт, визначають успіх продукту на ринку.

Основні функціональні вимоги до інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирному будинку:

- Моніторинг споживання: програмне забезпечення систематично обробляє дані споживання електроенергії в реальному часі, отримані від Smart Box пристроїв, забезпечуючи точне та актуальне відображення енергетичного профілю кожної квартири.
- Збереження даних: всі зібрані та оброблені дані систематично записуються в базу даних (або інше місце) для створення історичного архіву, що використовується для аналітики, звітності та стратегічного планування.
- Віддалене управління: при необхідності, система має можливість дистанційно відключити або підключити електроенергію до окремих квартир або будинку в цілому. Даний функціонал надає через інтерфейс IoT Gateway.
- Автоматичне реагування на перевантаження: в разі виявлення або наближення до критичної точки навантаження на генератор, система автоматично вживає заходів для запобігання відключення всього будинку від електромережі, використовуючи для цього систему сповіщення та силове відключення в критичному випадку.

  Основні нефункціональні вимоги:

  - Доступність (availability): система доступна для користувачів більшу частину часу, мінімізуючи періоди простою.
  - Розширюваність (extensibility): архітектура системи дозволяє легке додавання нового функціоналу або інтеграцію з іншими системами без значного перепроектування існуючої структури.
  - Спостережливість (observability): ключова характеристика хмарного сервісу — спостережливість, яка дозволяє розробникам та адміністраторам системи відстежувати, аналізувати та розуміти поточний стан системи та її історичну поведінку для швидкого виявлення та вирішення проблем.
  - Масштабованість (scalability): система підтримує зростання кількості даних та користувачів без втрати продуктивності.

- Відмовостійкість (fault tolerance): система здатна продовжувати роботу навіть у випадку часткових збоїв, з мінімальним впливом на загальну функціональність.

2.3.2 Мікросервіси та SOA

Архітектура орієнтована на сервіси (**Service-Oriented Architecture**) та мікросервісна архітектура [13] є двома важливими поняттями в сучасному проектуванні програмних систем, які забезпечують гнучкість та масштабованість в обробці даних і наданні послуг.

Сервіс-орієнтована архітектура (SOA) [13] - це архітектурний патерн, який дозволяє використовувати розподілені шматки програмного забезпечення, відомі як сервіси, що взаємодіють один з одним. Історія SOA сягає корінням у 1990-ті роки, коли компанії почали шукати способи зробити свої IT-системи більш адаптивними до швидких змін у бізнесі. Головна ідея полягала в створенні високої взаємодії між окремими компонентами програмного забезпечення, що могли б функціонувати незалежно один від одного.

Основні властивості SOA включають:

- Відділення функціоналу: Кожен сервіс в SOA є самодостатнім і реалізує певний бізнес-функціонал.
- Стандартизовані інтерфейси: Сервіси спілкуються через стандартизовані протоколи і інтерфейси, які визначають, як можна викликати функції сервісу.
- Легкість інтеграції: SOA сприяє інтеграції різноманітних технологічних рішень, навіть якщо вони реалізовані на різних платформах або мовах програмування.

На рисунку 2.9 проілюстровано основні елементи архітектури.

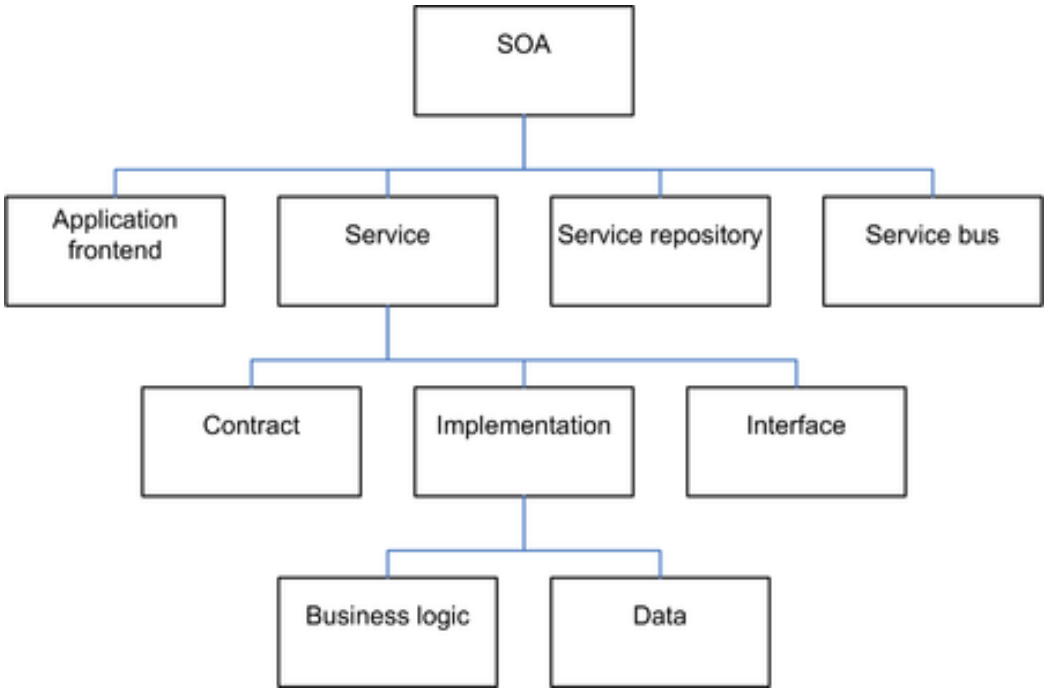

Рисунок 2.9 - Елементи SOA [13]

Архітектура, орієнтована на сервіси (SOA), пропонує рішення для цілого ряду обмежень, притаманних монолітній архітектурі. В монолітних системах розробники пишуть код для всіх сервісних функцій у рамках єдиної кодової бази, що може призвести до низки складнощів:

- Проблеми масштабування: одним з найбільших викликів є необхідність масштабування всього додатку, навіть коли лише окремий компонент потребує додаткових ресурсів. SOA дозволяє масштабувати кожен сервіс незалежно, що забезпечує більш ефективне використання ресурсів.
- Негнучке додавання чи зміни функціоналу: у монолітній архітектурі функціональність є розподіленою по всій кодовій базі, що ускладнює її модифікацію або розширення. SOA дозволяє легко додавати чи модифікувати сервіси без необхідності змінювати весь додаток.
- Неможливість повторного використання компонентів: В монолітних системах компоненти часто є неповторно використаними, у той час як SOA сприяє створенню взаємозамінних та повторно використовуваних сервісів.

- Обмежена відмовостійкість: якщо один компонент монолітної системи виходить з ладу, це може призвести до збоїв у всій системі. SOA підвищує відмовостійкість, ізолюючи помилки в окремих сервісах.
- Проблеми з впровадженням нових технологій та інтеграцією: Монолітні архітектури часто мають обмеження щодо використання нових технологій та інтеграції з зовнішніми системами, SOA ж надає гнучкість у виборі технологій та спрощує інтеграцію з іншими системами.
- Централізація відповідальності: В монолітних системах одна команда розробників часто відповідає за весь додаток, що може створювати проблеми для неперервної доставки та практик DevOps.

SOA надає розробникам можливість розбивати функціональні можливості програмного забезпечення на рівні постачальників і споживачів сервісів, які взаємодіють і обмінюються даними через корпоративну сервісну шину (ESB). Ця модель дозволяє спростити складні застосунки на декілька повторно використовуваних сервісів і робить розробку більш ефективною. Більш наглядно про користь SOA підходу показано на рисунку 2.10.

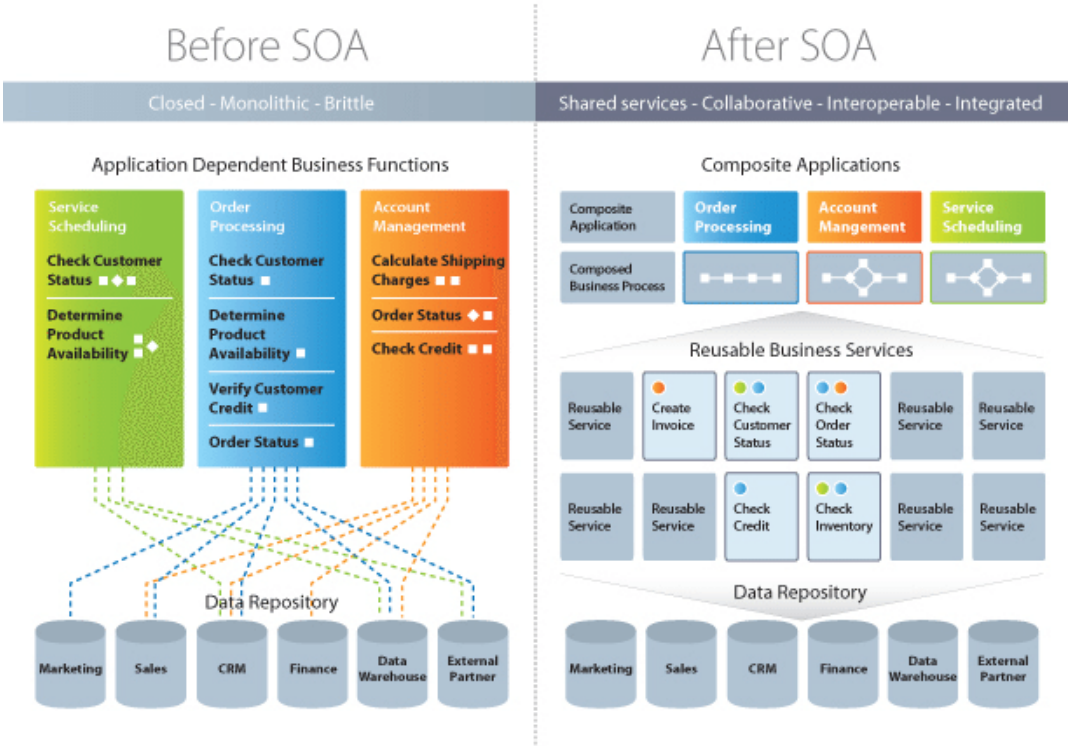

Рисунок 2.10 - Перевага SOA над монолітом [14]

Архітектура мікросервісів [15] є еволюцією стилю архітектури SOA. Якщо кожен сервіс в SOA є повноцінною бізнес-можливістю, то кожен мікросервіс являє собою значно менший програмний компонент, який спеціалізується лише на одному завданні. Мікросервіси усувають недоліки SOA та роблять програмне забезпечення більш сумісним з сучасними хмарними корпоративними середовищами. Це шлях розбиття великої програми на слабо пов'язані модулі (приклад на рисунку 2.11), які комунікують один з одним за допомогою просто API.

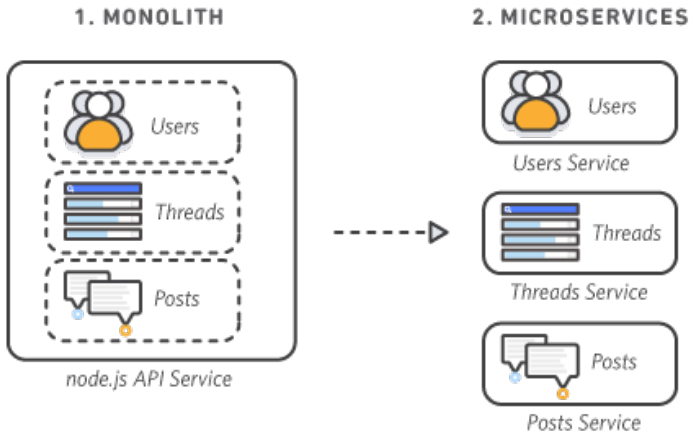

Рисунок 2.11 - Розбиття моноліту на мікросервіси [13]

Термін "архітектура мікросервісів", який також відомий як "мікросервіси", виник у середині 2010-х років і описує певний стиль архітектури в процесі розробки програмного забезпечення [15]. Особливий акцент на цьому стилі розробки зростав з часом, паралельно з еволюцією практик гнучкої розробки та DevOps, ставши відомим завдяки численним публікаціям, обговоренням у фахових спільнотах та виступам на профільних конференціях. В контексті динамічного розвитку та впровадження складних корпоративних додатків, архітектура мікросервісів демонструє значні переваги.

Архітектурний стиль мікросервісів полягає у створенні єдиного додатка як набору малих, самодостатніх та взаємонезалежних сервісів, які комунікують між собою за допомогою легковагових механізмів, таких як HTTP, AMQP, gRPC. Кожен сервіс орієнтований на окремий бізнес-процес та може бути розгорнутий незалежно у повністю автоматизованому середовищі, мінімізуючи

централізоване управління. Сервіси можуть бути розроблені на різних мовах програмування та використовувати різні технології зберігання даних.

Основна причина застосування мікросервісів — бажання компаній до швидкого адаптування під зміни бізнес-вимог, що дозволяє залишатися на крок попереду конкурентів. Мікросервіси полегшують внесення змін, забезпечуючи високу частоту оновлень без значного впливу на загальну структуру системи.

Однак, слід пам'ятати, що цей підхід вносить додаткову складність у проект. Вимагається належний рівень співпраці та комунікації між командами DevOps і розробниками, а також ретельно продумана система моніторингу та виявлення збоїв. Тому сильна DevOps-культура в організації є критично важливою для успішного застосування мікросервісів.

Мікросервіси vs моноліт. З метою глибшого розуміння сутності мікросервісної архітектури важливо провести порівняльний аналіз із класичним підходом, що відомий як монолітна архітектура.

Традиційно, монолітні рішення вибудовуються як компактні єдності, де будь-яке оновлення або зміна, незалежно від їх масштабу, вимагає повного перезапуску і розгортання всієї системи. Такий підхід із часом ускладнює підтримку чіткої модульної структури, а зміни в одному компоненті часто порушують функціонал інших. Крім того, масштабування монолітних застосунків може стати викликом, оскільки потреби в ресурсах різних модулів часто конфліктують між собою. Отже, необхідно знайти компроміс між вимогами до обладнання та фактичними можливостями системи.

Мікросервісна архітектура запропонувала рішення цих проблем шляхом незалежного розгортання кожного сервісу. Зміни в одному сервісі не впливають на роботу інших, що дозволяє ефективніше управляти окремими елементами системи.

Разом із перевагами, мікросервісна архітектура також представляє ряд викликів:

- Мережеві затримки через необхідність комунікації між атомізованими сервісами через мережу.

- Стандартизація форматів повідомлень і потенційні помилки у налагодженні зв'язку між сервісами.
- Налагодження балансування навантаження і забезпечення відмовостійкості системи.
- Операційна складність, що вимагає відповідної кваліфікації DevOps-фахівців, а також впровадження неперервного розгортання і моніторинг системи.
- Проблематика тестування, оскільки мікросервіси вимагають індивідуального запуску та перевірки перед повноцінним інтеграційним тестуванням.

Отже, в контексті сучасного програмування, архітектура мікросервісів є одним з передових підходів до структурування додатків. Вона втілює в собі велику гнучкість, дозволяючи окремим сервісам працювати з підвищеною швидкодією, що сприяє зростанню продуктивності розробників та оптимізації процесів розгортання.

Основні переваги, які надає цей підхід, охоплюють:

- Більшу ефективність запуску та розгортання сервісів, що зумовлено їхньою автономністю.
- Незалежність у розгортанні, яка дозволяє вносити зміни в індивідуальні сервіси без впливу на решту системи.
- Гнучке масштабування кожного сервісу за потребою.
- Посилена стійкість до збоїв, адже проблеми в одному мікросервісі рідко призводять до глобальних неполадок у системі.
- Відсутність тривалих технологічних зобов’язань, що надає свободу у виборі технічного стеку для кожного нового сервісу.
- Легкість у реорганізації та налаштуванні сервісів для виконання нових завдань.

Проте, варто зважити і на недоліки, пов’язані з мікросервісами:

- Додаткова складність розробки розподілених систем, що вимагає від розробників вищого рівня кваліфікації та усвідомлення особливостей таких систем.
- Оперативні складнощі управління та розгортання системи з безлічі сервісів.
- Непередбачувані виклики, з якими можуть зіткнутися розробники при створенні нової мікросервісної архітектури, включаючи несподівані проблеми на різних рівнях взаємодії.

На рисунку 2.12 зображано 8 «оман розподілених систем» [15], в які дуже легко впасти при розробці мікросервісної архітектури. Цей перелік також можна віднести до недоліків такого підходу.

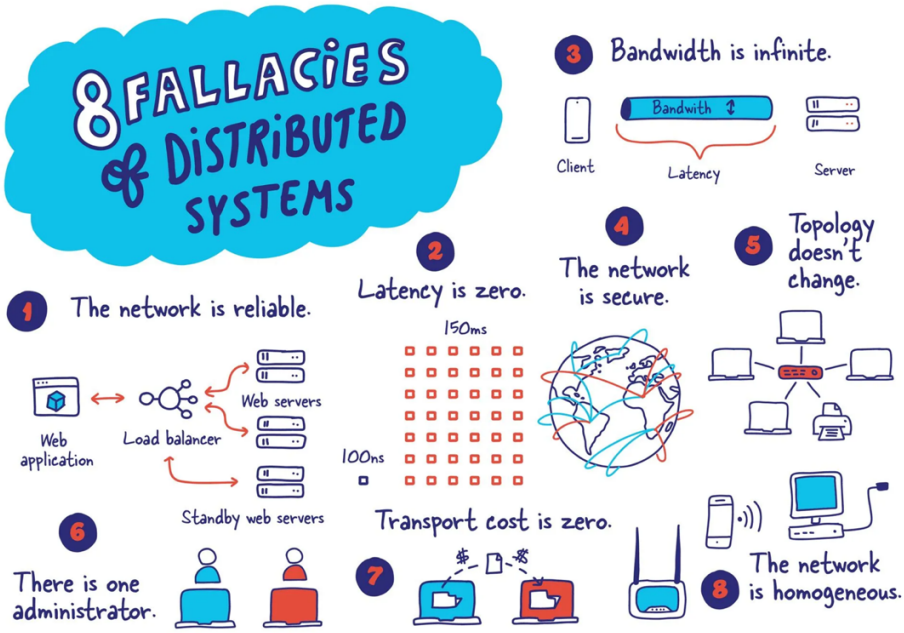

Рисунок 2.12 - Вісім хибних тверджень для розподілених систем [15]

У світі IT велика кількість лідерів ринку відмовилася від традиційних монолітних архітектур на користь більш модульних та адаптивних підходів. Цей перехід дозволив їм досягти нових висот у швидкості та гнучкості розробки продуктів. Серед таких компаній можна відзначити таких гігантів як Amazon та eBay, що стали піонерами у використанні мікросервісів, а також Walmart і Netflix, які відомі своєю інноваційністю в цифровій комерції та стрімінгових сервісах відповідно. До цього списку варто додати SoundCloud та Spotify -

компанії, що змінили підхід до розповсюдження музики, Twitter з його соціальною мережею, Stripe і PayPal, які перетворили сферу онлайн-платежів, а також Uber, який революціонізував пасажирські перевезення, та Medium - платформа для блогерів та письменників. Кожна з цих компаній використовує мікросервісну архітектуру, що дозволяє їм бути більш відзивчивими до змін ринку та швидше адаптуватися до потреб користувачів.

Експеримент Netflix з мікросервісною архітектурою виявився настільки результативним, що компанія навіть вирішила надати вільний доступ до ряду розроблених ними програмних інструментів. Ці інструменти сприяли формуванню їхньої унікальної архітектури мікросервісів. Завдяки такому кроку, Netflix став своєрідним еталоном у галузі впровадження мікросервісів, ставши предметом досліджень та наслідування численними компаніями у всьому світі. Їхній підхід та інноваційні рішення надихнули багатьох на використання мікросервісів у своїх проектах, сприяючи розвитку цього напрямку в IT індустрії. Нижче проілюстровано приблизний дизайн бекенд архітектури Netflix.

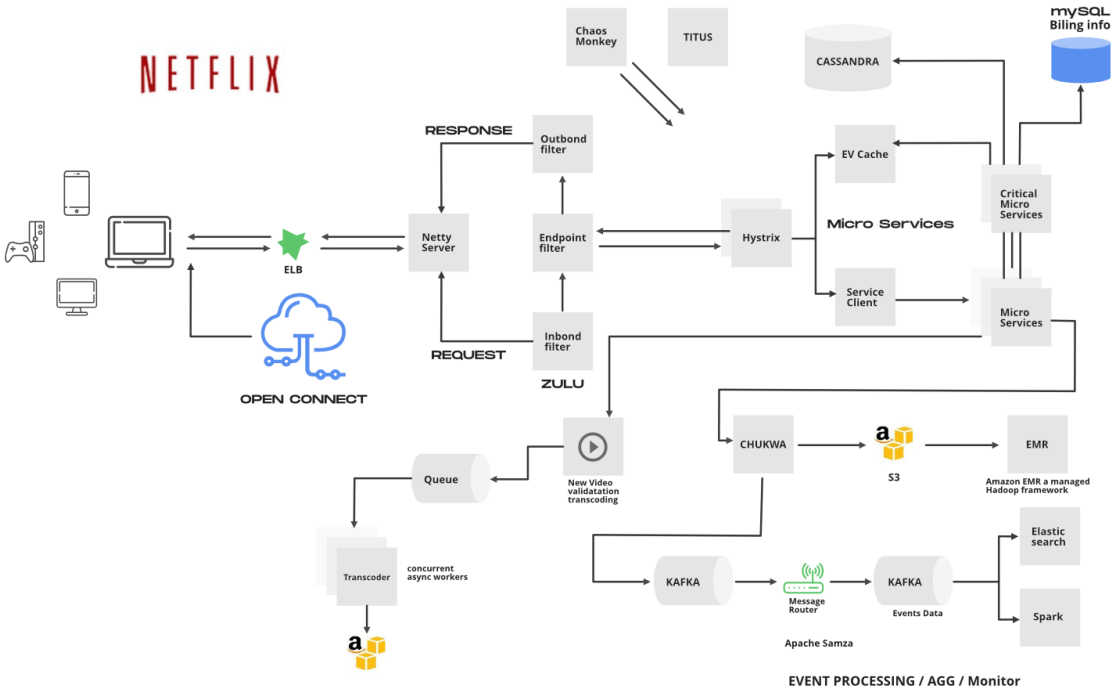

Рисунок 2.13 - Мікросервіси Netflix [16]

2.3.3 Розробка архітектури програми

На рисунку 2.14 представлено System Design програмної частини роботи.

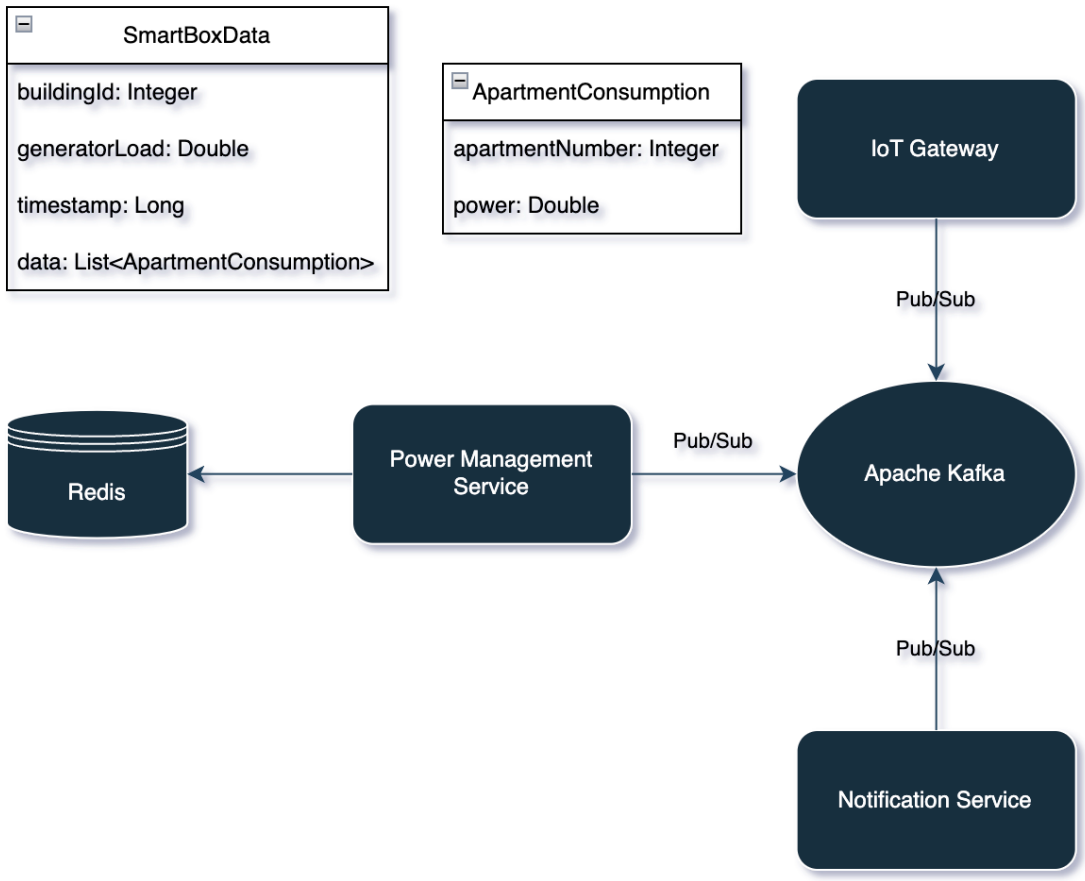

Рисунок 2.14 - System Design

Як бачимо, програмне забезпечення розподіленої системи контролю споживання електроенергії в багатоквартирному будинку на даний момент складається з п’яти компонентів.

IoT Gateway сервіс. Його апаратну частину було згадано в розділі опису технічної сторони. Це сервіс, що відповідає за комунікацію пристроїв, що знаходяться в будинку з програмним забезпеченням, що знаходиться в хмарі та до якого є доступ по мережі інтернет. Хочу зазначити, що задля тестування роботи алгоритму та системи в цілому, було реалізовано емуляцію реальних даних споживання від квартир за допомогою засобів програмування (детальніше про це – у розділі реалізації).

Power Management сервіс. Безпосередньо основний сервіс, який включає в собі реалізацію розробленого алгоритму, за яким працює вся система на випадок живлення від резервного джерела електроенергії. Найбільший сервіс із запропонованих вище. Такий сервіс повинен зберігати стан будинку та квартир в ньому, всю необхідну інформацію для коректної роботи алгоритму та деякі конфігураційні дані – тому до поточного сервісу підключається **Redis**, з обґрунтуванням вибору якого можна ознайомитись у відповідному розділі в цій дисертації.

Notification сервіс. Частина що відповідає за надсилання повідомлень кінцевим користувачам – мешканцям квартир. Одна із ключових задач алгоритму – вчасне та інформативне сповіщення людей про чимало подій, що відбуваються під час роботи алгоритму: відключення світла, перенавантаження генератора, порушення правил обмеження споживання та ін. В даній роботі в якості зв’язку із споживачами використовується email пошта.

І останній компонент, один із найважливіших, що відповідає за комунікацію та рух даних системи в цілому, що виступає інтерфейсом між IoT Gateway сервісом та хмарним застосунком – **Apache Kafka** [17]**.** Змістовний опис та обґрунтування цього вибору також представлено в одному з наступних розділів дисертації.

Крім цього, на рисунку можна побачити опис моделі даних, що IoT Gateway збирає та передає кожну секунду від Smart Box пристроїв до Apache Kafka.

2.3.4 Вибір системи обміну повідомленнями

Розмови про взаємодію мікросервісів часто розпочинається з розгляду REST API, зокрема RESTful веб-систем. Цей підхід є вельми популярним через його простоту та легкість у реалізації. Однак, REST API має деякі обмеження:

- REST API спочатку було розроблено для обробки команд та запитів, але не для передачі подій або нотифікацій.

- Слабка гнучкість в інтеграції: сервіс, що надсилає запит, мусить знати одержувача заздалегідь, що призводить до тісного зв'язку між відправником і одержувачем.
- Ускладнення гарантованої доставки повідомлення у випадку недоступності або перевантаженості одержувача.
- Збільшення загальної затримки (latency) через необхідність очікування на завершення обробки запиту.

Альтернативний підхід передбачає використання структури даних, схожої на чергу, але з розширеними можливостями. Така черга має зберігати дані поза JVM, на дисковій підсистемі, забезпечувати високу доступність і масштабованість. Існують різні технології і платформи, які можна використовувати у таких цілях, наприклад:

- Сервіси повідомлення в черзі (наприклад, StormMQ, Amazon SQS).
- Посередники орієнтовані на повідомлення (наприклад, RabbitMQ).
- Обробка потоків даних або подій (наприклад, Apache Kafka). На рисунку 2.15 представлено приклад потокової передачі даних в реальному часі.

Ці системи мають складну архітектуру та використовують різноманітні математичні алгоритми для обробки та передачі даних. Зазвичай їх розробка базується на вже існуючих рішеннях та використовує відомі патерни, замість створення з нуля. У загальному вони відомі як системи обміну повідомленнями.

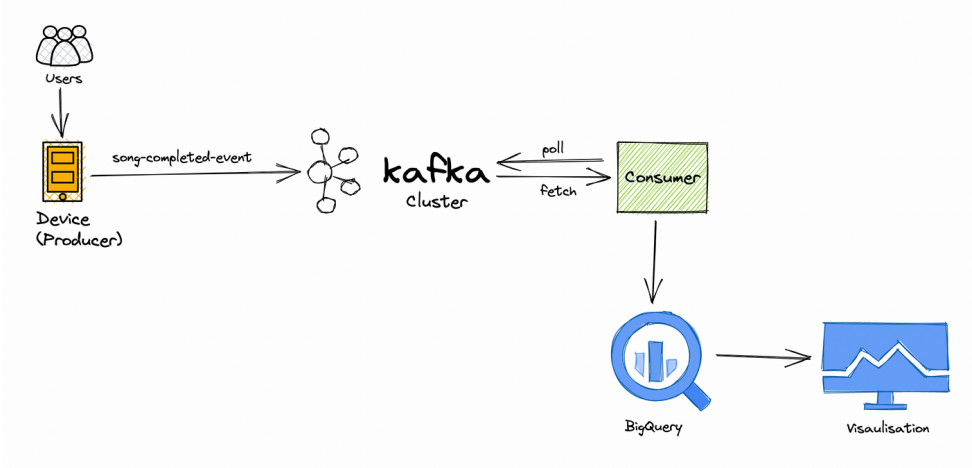

Рисунок 2.15 - Real-time streaming architecture [17]

Цей розділ приділяє увагу Kafka, але перед глибшим аналізом цієї системи варто розглянути інші рішення для обміну повідомленнями та потокової обробки подій, які конкурують з Kafka або були її попередниками. Kafka є складною у налаштуванні та управлінні системою, і не завжди є оптимальним рішенням. Аналіз альтернатив дозволить краще зрозуміти, в яких випадках варто вибирати Kafka, а в яких - інші системи обміну повідомленнями.

Оглянемо системи, які з'явились на ринку до Kafka [17]:

ZeroMQ (рисунок 2.16): Ця бібліотека з’явилася у 2007 році, спочатку була написана на C++, а потім адаптована під різні мови програмування. Головна особливість ZeroMQ полягає в тому, що це бібліотека, а не окремий сервер, і вона не потребує виділеного сервера для зберігання повідомлень. Вона працює на основі TCP/IP і сокетів, надаючи можливості асинхронної доставки повідомлень і підтримуючи патерни взаємодії "запит/відповідь" та "публікація/підписка".

Рисунок 2.16 - ZeroMQ [17]

ActiveMQ (рисунок 2.17): Цей проєкт Apache Foundation, який з’явився у 2004 році, є повноцінним брокером повідомлень, який реалізує JMS 1.1. ActiveMQ пропонує зберігання даних на диску чи в базі даних через JDBC, підтримує кластеризацію та реплікацію, а також інтеграцію з різними мовами програмування через універсальні протоколи.

Рисунок 2.17 - ActiveMQ [17]

RabbitMQ (рисунок 2.18): Створена у 2007 році та відома своєю потужністю, RabbitMQ написана на Erlang і пропонує клієнти для багатьох мов програмування. Ця система позиціонує себе як брокер повідомлень із можливістю розширення через плагіни, підтримує сучасні протоколи та кластеризацію. RabbitMQ забезпечує гнучкість у маршрутизації повідомлень.

Рисунок 2.18 - RabbitMQ [17]

Для обміну повідомленнями ці системи використовують різні протоколи і підходи, що робить їх адаптованими до різноманітних потреб. Водночас, специфікація JMS визначає основні компоненти систем обміну повідомленнями, але не всі ці системи повністю підтримують її, оскільки JMS розвивався повільно, що спонукало до розробки альтернативних технологій.

Apache Kafka можна порівняти з молодшим членом великої родини технологій обробки повідомлень, який з'явився на світло, коли ринок вже був насичений основними гравцями. Однак, Kafka вирішила знайти своє місце, ставши високопродуктивною платформою для потокової обробки подій. Офіційно Kafka не вважається звичайним брокером повідомлень, а представляє собою систему для потокової обробки подій, використовуючи термінологію "подія" замість "повідомлення".

Розробка Kafka розпочалася в 2009 році в LinkedIn на Java і Scala. У 2011 році Kafka перетворилася на відкрите програмне забезпечення, а у 2014 році один із її основних розробників, Джей Крепс, заснував Confluent [18]. Ця компанія працює над доповненням Kafka різними корпоративними функціями для широкого спектру клієнтів. Таким чином, Kafka доступна у базовому вигляді як Apache Kafka, а також у вдосконаленому варіанті від Confluent, який

пропонується у вільній та комерційній версіях, або ж wurstmeister, що і було використано в даній роботі.

Основні причини популярності Apache Kafka:

- Розподілена архітектура з підтримкою реплікації даних.
- Висока продуктивність обробки даних завдяки розподілу черг на розділи (partitions), що дозволяє паралельну обробку.
- Надійне зберігання повідомлень на диску, не видаляючи їх після обробки.
- Гарантована доставка та отримання повідомлень.
- Підтримка клієнтів для багатьох мов програмування.
- Придатність для систем реального часу.

Kafka є невід'ємною частиною так званого SMACK стека (Spark, Mesos, Akka, Cassandra, Kafka) [19], який вважається стандартом для обробки великих обсягів даних. В той же час, Kafka технологічно складна, оскільки використовує Apache Zookeeper для управління конфігурацією та кластеризації. Це означає, що для забезпечення високої доступності та відмовостійкості потрібно налаштувати кластер, що включає як сервери Kafka, так і Zookeeper.

Зберігання даних. У Apache Kafka основним елементом є повідомлення, яке виконує роль аналогічну запису в базі даних. Всі операції і сервіси Kafka базуються на обробці повідомлень. Структура повідомлення включає в себе наступні компоненти [17]:

- Ключ.
- Значення.
- Час відправлення (timestamp).
- Метадані (заголовки), які з’явилися у версії Kafka 0.11.

Ключ та значення представляють собою масиви байтів, причому Kafka не вимагає специфічного формату для значень, дозволяючи використовувати JSON, Avro, Protobuf тощо. Обов'язковими є лише значення та timestamp. Такий підхід відрізняється від реляційних баз даних, де кожен запис має унікальний первинний ключ. У Kafka, унікальним ідентифікатором для запису є його offset

- позиція в черзі. Ключ важливий для споживачів, оскільки він логічно вказує на приналежність повідомлення до конкретного об'єкту.

При надходженні повідомлення на брокер Kafka, воно розміщується у topic (ілюстрація на рисунку 2.19).

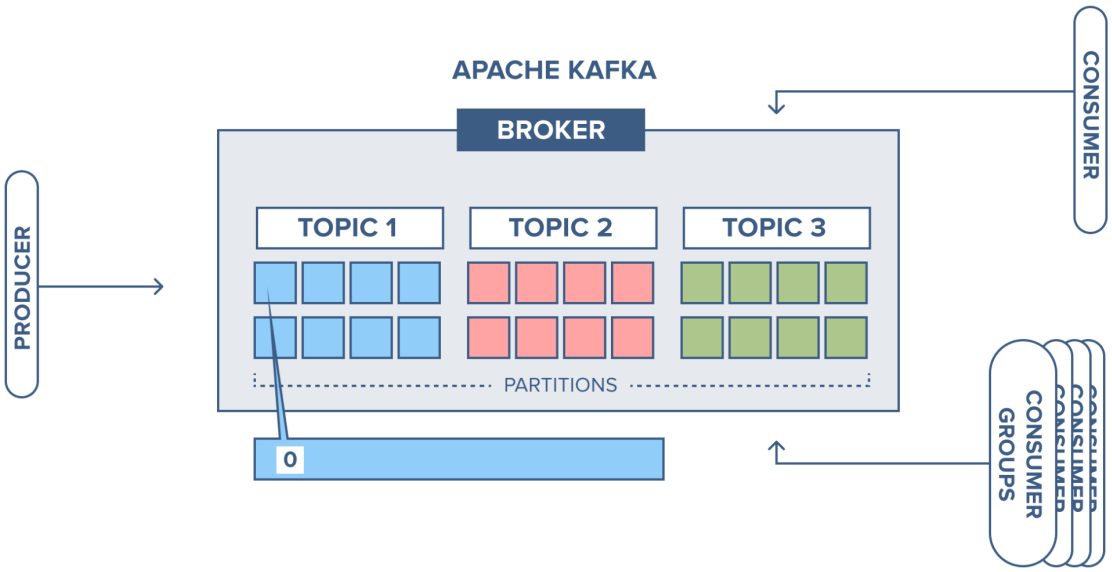

Рисунок 2.19 - Структура Кафки [17]

Topic в Kafka - це аналог папки, яка логічно об'єднує повідомлення за певною функціональною ознакою. Відправники і споживачі повинні домовитися про використання певного topic, вказавши його назву, яка має бути унікальною в рамках брокера. Topic має існувати на момент відправлення повідомлення, або властивість auto.create.topics.enable має бути встановлена як true. Такий підхід також існує в MongoDB, де колекція створюється автоматично при першому зверненні до неї.

Topics у Kafka можуть бути уявлені аналогічно до таблиць у базах даних, але з певними відмінностями: у випадку реляційної СУБД кожен запис має єдиний формат, тоді як у NoSQL базах даних документи можуть мати різноманітну структуру. У Kafka існує баланс між цими підходами. Наприклад, якщо існує багато подій, пов'язаних з однією сферою, такою як операції з платежами, можливе створення окремих topics для кожного типу подій. Це спрощує обробку, але не гарантує послідовність їх обробки, створюючи потенційні ризики для колізій.

Topics у Kafka не є однорідними; вони поділяються на розділи (**partitions**), які не дублюють повідомлення між собою. Ця структура, схожа на бакети у HashMap, призначена для оптимізації процесу доставки повідомлень споживачам. Розділення topics на розділи дозволяє асоціювати кожен розділ з окремим будинком для паралельної обробки. Kafka забезпечує, що повідомлення з однаковими ключами потрапляють у той самий розділ.

Проте, це може призвести до нерівномірного розподілу повідомлень. Наприклад, використання унікального ідентифікатора користувача як ключа може спричинити перевантаження деяких розділів. Генерування випадкового числа (UUID) як ключа розв'язує цю проблему, але в такому випадку повідомлення одного користувача можуть опинитися у різних розділах, що порушує послідовність їх обробки. Таким чином, якщо послідовність обробки повідомлень має важливе значення, необхідно повернутися до попереднього варіанту, де повідомлення одного типу зберігаються у одному topic.

В даній роботі використано саме попередній принцип – один topic на один тип повідомлення. До того ж, всі Smart Box дані від одного будинку попадуть до одного того ж самого розділу, адже послідовність подій дуже важлива в нашому випадку і ми не можемо цим нехтувати.

Розділ у Kafka фізично являє собою лог-файл (ілюстрація на рисунку 2.20), в якому зберігаються повідомлення. Через велику кількість повідомлень, які можуть становити сотні тисяч або навіть мільйони, Kafka поділяє цей лог-файл на дрібніші сегменти фіксованого розміру для оптимізації обробки даних та управління ними. Коли сегмент досягає свого максимального розміру, він закривається, і створюється новий порожній сегмент для подальшого зберігання повідомлень.

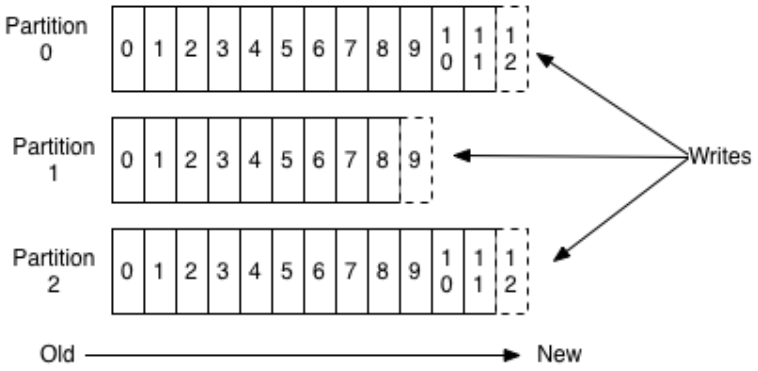

Рисунок 2.20 - Анатомія топіку [19]

Для ефективної взаємодії через Kafka, відправники та споживачі повинні домовитися заздалегідь про кілька ключових аспектів: назву topic'а, тип та наявність ключа в повідомленні, і формат зберігання даних у повідомленні, такий як JSON або Protobuf.

Kafka є унікальною комбінацією властивостей системи передачі повідомлень та бази даних. На відміну від традиційних баз даних, де дані зберігаються на тривалий час, Kafka впроваджує політику "retention", тобто автоматичного очищення старих повідомлень в залежності від встановлених лімітів за кількістю або часом зберігання. Крім того, Kafka може стиснути дані у розділах, видаляючи старі повідомлення та елімінувати дублікати на основі ключів, зменшуючи обсяг зберігання і забезпечуючи оптимізацію ресурсів.

Отже, основною перевагою Kafka є її здатність до швидкої і масштабованої передачі повідомлень, що є критично важливим для систем, де необхідна гарантована доставка повідомлень та їх швидка обробка. Використання Kafka в IoT-проектах має особливе значення, оскільки це дозволяє ефективно обробляти великі потоки даних, які генеруються безліччю підключених пристроїв.

В IoT-технологіях Kafka використовується для збору та обробки даних від датчиків, які постійно генерують великі масиви інформації. Дані з датчиків можуть включати показники температури, вологості, руху та інші вимірювання, які необхідно швидко обробляти та аналізувати, в нашому випадку – це електроспоживання. Kafka забезпечує миттєве відправлення та обробку цих даних, що дозволяє системам реагувати в реальному часі.

Також Kafka ефективна у вирішенні завдань, пов'язаних з аналізом даних та машинним навчанням. Великі обсяги даних, зібрані з IoT-пристроїв, можуть бути використані для аналітичних цілей та підтримки прийняття рішень, що є важливим аспектом розвитку інтелектуальних систем.

Отже, використання Apache Kafka у сфері IoT-проектів та мікросервісних архітектур стало важливим рішенням для ефективного оброблення великих потоків даних та реалізації вимог сучасних технологічних систем. Kafka не тільки підвищує продуктивність та надійність цих систем, але й вносить важливий вклад у розвиток сфери IoT, забезпечуючи оптимальне використання та аналіз великих масивів даних.

2.3.5 Вибір мови програмування та фреймворку

Java [20] є однією з найпопулярніших мов програмування в світі, особливо коли йдеться про розробку серверних додатків для корпоративних систем. Вона поєднує в собі низку характеристик та переваг, які роблять її ідеальною для розробки нашої системи контролю споживання електроенергії в багатоквартирних будинках.

- Переносимість та платформонезалежність: Java-додатки можна виконувати на будь-якій платформі без змін у коді завдяки віртуальній машині Java (JVM). Це робить Java ідеальною для систем, які потребують високої ступеня переносимості.
- Висока продуктивність: Завдяки оптимізаціям JVM, Java забезпечує високу продуктивність, що є критично важливим для обробки великих обсягів даних в системах контролю споживання електроенергії.
- Масштабованість і надійність: Java має потужну систему управління пам'яттю та вбудовані засоби для масштабування, які дозволяють розробляти надійні та стабільні системи, що можуть ефективно обслуговувати велику кількість користувачів та датчиків.

- Широкий Набір Бібліотек та Інструментів: Java пропонує величезний набір бібліотек та інструментів, що дозволяє розробникам використовувати вже готові рішення для створення різноманітних частин системи.
- Сильне Співтовариство та Підтримка: Як одна з найстаріших та найбільш використовуваних мов програмування, Java має міцне співтовариство розробників та широку підтримку, яка може бути корисною для вирішення проблем та підтримки системи.
- Безпека: Java надає міцні механізми безпеки, які є важливими для систем контролю споживання електроенергії, особливо коли йдеться про захист даних користувачів.
- Зрілість та Стабільність: Java є зрілою мовою з визначеним життєвим циклом розробки та стабільними версіями, що забезпечує впевненість у виборі технології для довгострокових проектів.

Вибір Java для нашої системи обумовлений її зрілістю, високою продуктивністю, масштабованістю, багатою екосистемою та підтримкою багатопоточності. Ці якості роблять Java надійним вибором для розробки розподіленої системи контролю споживання електроенергії.

Spring [21] – один із найпопулярніших фреймворків для створення веб-додатків. Нижче наведено основні причини вибору саме цього фреймворку для реалізації інтелектуальної розподіленої систему контролю споживання електроенергії в багатоквартирному будинку.

- Відповідність мові програмування. Інтеграція з Java - вибір Java як основної мови програмування для системи значною мірою обумовлює вибір Spring. Spring є одним з найпопулярніших та найбільш зрілих Java фреймворків, який пропонує широкий спектр можливостей для розробки.
- Принципи та Парадигми Розробки.
    - Convention Over Configuration. Spring спрощує конфігурацію, використовуючи звичні шаблони та конвенції, що знижує кількість необхідного коду та помилок.

- Inversion of Control (IoC) та Dependency Injection. Ці ключові концепції Spring дозволяють зменшити залежності між компонентами системи, сприяючи їх легшому тестуванню та більшій гнучкості.

- Підтримка мікросервісної архітектури.
  - Spring Boot та Spring Cloud. Ці проекти Spring спеціально призначені для спрощення розробки мікросервісів, надаючи швидке створення стендових рішень та підтримку хмарних сервісів.
- Широка екосистема та інтеграція.
  - Підтримка різних модулів. Spring пропонує модулі для різних завдань, включаючи доступ до даних (Spring Data), веб-розробку (Spring MVC), безпеку (Spring Security), і багато іншого.
  - Інтеграція з Іншими Технологіями. Spring легко інтегрується з іншими технологіями та фреймворками, що дозволяє гнучко підходити до розробки різноманітних компонентів системи.
- Спрощення розробки та підтримки.
  - Спрощення конфігурації та управління. Інструменти Spring, такі як Spring Boot, забезпечують автоматизовану конфігурацію та управління залежностями, знижуючи витрати часу на ці процеси.
  - Покращення Продуктивності Розробників: Spring сприяє швидкій розробці високоякісних додатків за рахунок своєї широкої функціональності та легкості використання.

Узагальнюючи, вибір Spring як фреймворку для розробки розподіленої системи контролю споживання електроенергії забезпечує гнучкість, швидкість розробки та високий рівень інтеграції з Java, що є ключовими аспектами для досягнення ефективності та надійності в даному проекті.

2.3.6 Вибір бази даних

Вибір бази даних для системи керування споживанням електроенергії є критичним, оскільки вона безпосередньо впливає на продуктивність, масштабованість та надійність системи. **Redis** (ілюстрація 2.21) [22] було

вибрано як основну базу даних для Power Management Service через його високу швидкість та ефективність у роботі з даними, які часто оновлюються та запитуються, що є типовим для IoT-орієнтованих систем.

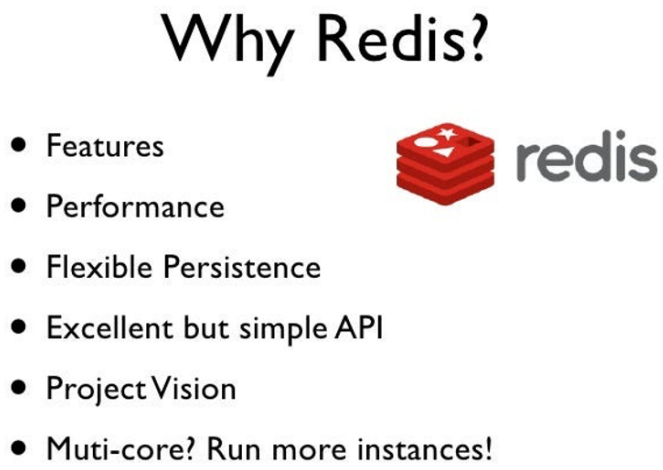

Рисунок 2.21 - «Чому Redis?» [22]

Redis, що є в пам'яті структурованим сховищем даних типу "ключ-значення", вирізняється своєю високою продуктивністю в операціях читання та запису, що дозволяє миттєво отримувати та оновлювати стани квартир та будинку. Це особливо важливо для миттєвої реакції на зміни в навантаженні та управлінні споживанням електроенергії в реальному часі.

Порівняно з реляційними базами даних, такими як PostgreSQL [23], Redis пропонує більш швидкий доступ до даних через використання пам'яті замість дискових операцій, що є критичним для забезпечення високої пропускної спроможності системи. Хоча PostgreSQL є відмінним варіантом для комплексних запитів та транзакцій, її використання може бути надмірним для сценаріїв, що вимагають швидкого відгуку на події в реальному часі, як у випадку з нашою системою.

З іншого боку, MongoDB [24] як документоорієнтована база даних також надає гнучкість у структуризації даних і можливість зберігання JSON-подібних документів. Однак, в контексті IoT-проектів, де частота записів та запитів є високою, Redis все одно вважається більш підходящим, оскільки він забезпечує менші затримки та краще відповідає потребам в системах, що вимагають швидкодії.

Редіс також підтримує різноманітні структури даних, такі як списки, множини, сортовані множини, хеші, що дозволяє гнучко управляти складними даними, які містять ієрархічні та мультимодальні зв'язки, типові для багатоквартирних будинків та їх споживачів.

Зазначимо також, що Redis має вбудовані можливості для реплікації та підтримки високої доступності, що є важливим для критичних систем, де вимога до безперервності сервісу є високою.

У підсумку, вибір Redis для Power Management Service визначається його високою продуктивністю, гнучкістю у роботі з даними, широкими можливостями масштабування та надійністю у забезпеченні безперервної роботи системи.

2.4 Алгоритм роботи системи

Основний предмет дослідження теоретичної складової даної роботи – це алгоритм, за яким буде працювати система, для того щоб досягти поставлених цілей – запобігти аварійному відключенню резервного джерела живлення та забезпечити максимальний комфорт мешканцям наскільки це можливо при аварійній ситуації. Практична реалізація даного алгоритму представлена в розділі «Реалізація системи».

Алгоритм роботи інтелектуальної системи контролю споживання електроенергії базується на скоординованому використанні даних та управлінні споживанням енергії в багатоквартирному будинку. Основоположним елементом є конфігураційна інформація про енергетичні характеристики будинку та всіх квартир, а також контактні дані мешканців для нотифікації. Всі вхідні дані зберігаються у конфігураційному файлі JSON, що зчитується при ініціалізації сервісу управління енергоспоживанням (Power Management Service).

Запуск системи починається з завантаження та обробки вказаних конфігураційних даних, після чого інформація переноситься до бази даних Redis. Зазначена база даних слугує як основний репозиторій для збереження станів будинку та квартир. Важливо відмітити, що алгоритм розроблено з можливістю масштабування для роботи з декількома будинками одночасно.

Як вже зазначалося, ключовим елементом роботи системи є Smart Box, який надсилає дані про споживання енергії з певною періодичністю в режимі Real-Time (за замовчуванням це раз на одну секунду). Система аналізує ці дані та реагує на події, такі як відключення від міської електромережі. У випадку такого відключення, система отримує сигнал про блекаут та АВР ініціює процес переходу на альтернативне джерело енергії — генератор.

Далі, сервіс Power Management розсилає нотифікації мешканцям про необхідність зниження споживання. Відповідно до останніх, отриманих безпосередньо перед блекаутом, даних, формується список квартир, що перевищують дозволені ліміти споживання. Ці квартири відразу відключаються від енергопостачання через IoT Gateway та Smart Box. Таким чином гарантуючи те, що генератор запуститься з першого разу та не буде перенавантажено з самого початку роботи.

Система встановлює чергу для підключення відключених квартир до живлення, враховуючи сигнали, отримані через "радіокнопку", яка свідчить про готовність мешканців дотримуватися режиму економії енергії. Квартири, які зазначили про свою готовність, отримують пріоритет у черзі на підключення.

Паралельно система моніторить поточне споживання та відключає квартири-порушники у разі ігнорування попереджень. У випадку перевантаження генератора, аварійно відключається енергопостачання, система очищує поточні дані та ініціює повторний процес підключення після відновлення генератора. У випадку відновлення міського енергопостачання, система нормалізує роботу, розсилаючи відповідні сповіщення та відновлюючи живлення всіх квартир.

Загалом, розроблений алгоритм забезпечує ефективне управління енергоспоживанням, оптимізує використання альтернативних джерел енергії та гарантує стабільну роботу житлового комплексу при аварійних ситуаціях на електромережі. До того ж, забезпечує максимально справедливе розподілення енергії та зворотній зв’язок з мешканцями – мінімізуються випадки

неочікуваного відключення, бо для всіх однаково надається деякий час на виправлення в разі порушень, а також надсилаються інформативні сповіщення.

Важливим є те, що даний алгоритм було ретельно пропрацьовано та представлено декільком інженерам-електрикам, які підтвердили його реалістичність, практичність та прикладну цінність.

2.5 Висновок

У даному розділі було детально розглянуто та проаналізовано ключові аспекти розробки інтелектуальної розподіленої системи контролю споживання електроенергії. В основу дослідження покладено розробку алгоритму, який виконує ряд критичних функцій, зокрема запобігання аварійному відключенню резервного джерела живлення та підтримку комфорту мешканців при аварійних ситуаціях.

Розроблений алгоритм, що базується на event-driven підході, враховує потреби реального часу та можливості масштабування системи для роботи з кількома будинками одночасно, забезпечуючи стабільне управління енергетичними ресурсами. Це дозволяє оптимізувати використання альтернативних джерел енергії, мінімізувати ризики перевантаження генератора та забезпечувати неперервну роботу критично важливих систем будинку.

Ефективність алгоритму засвідчена через моделювання різних сценаріїв, включаючи відключення від міської електромережі та автоматичне переключення на резервні джерела живлення. Система не тільки реагує на надзвичайні події, але й проактивно інформує користувачів про стан енергопостачання та необхідність зниження споживання, що допомагає уникнути непередбачуваних перебоїв у роботі.

Окремо слід виділити аспекти безпеки та надійності системи, які були ретельно продумані та реалізовані у алгоритмі. Відмічено, що Smart Box здатен функціонувати навіть під час відключення електроенергії, використовуючи власний акумуляторний блок для автономної роботи.

Вагомим є той факт, що алгоритм був оцінений декількома інженерами-електриками та професіоналом у галузі енергетики, які підтвердили його

практичну придатність та прикладну цінність. Відгуки експертів свідчать про високу ефективність та потенціал алгоритму для реалізації у великомасштабних житлових комплексах.

Алгоритм не тільки відповідає сучасним технологічним стандартам, але й враховує соціальну складову, сприяючи залученню мешканців до процесу управління енергоспоживанням. Така взаємодія між технологіями та користувачами створює фундамент для розбудови енергетично стійких та інтелектуальних житлових просторів майбутнього.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

Цей розділ присвячено деяким деталям реалізації сервісів та прикладам роботи системи в цілому.

Цей розділ присвячений більш детальному розгляду практичної сторони створеної системи, висвітленню ключових елементів програмування та конкретним прикладам функціонування розробленого програмного забезпечення. Значна увага приділяється аналізу нюансів і деталей, які виникають під час реалізації алгоритмів у вигляді окремих сервісів, з метою детальнішого розуміння механізмів роботи системи.

В контексті дослідження, реалізована інтелектуальна розподілена система є практичним застосуванням теоретичних знань і розроблених алгоритмів для керування енергоспоживанням в умовах відключення будинку від основної електромережі. Цей розділ відображає етап переходу від теорії до практики, показуючи, як теоретичні концепції втілюються у життя через програмний код та його функціонування в реальних умовах.

Представлений матеріал уникає зайвих подробиць, які не мають вирішального впливу на розуміння логіки системи, зосереджуючись на ключових аспектах реалізації. Деталі коду, які не відіграють важливої ролі в загальному функціонуванні системи, опущені з метою забезпечення концентрації на фундаментальних принципах та механізмах роботи програми.

Для всіх сервісів використовується єдине середовище розробки – Intellij Idea, що сприяє уніфікації процесу розробки та полегшує інтеграцію окремих компонентів системи.

3.1 IoT Gateway сервіс

Даний розділ призначений для вивчення структури та функціоналу сервісу IoT Gateway у межах розробленої системи. Задля демонстрації можливостей сервісу, акцент робиться на моделюванні взаємодії зі Smart Box, як ключового елемента в системі збору даних.

На рисунку 3.1 представлено структуру Spring Framework проекту IoT Gateway.

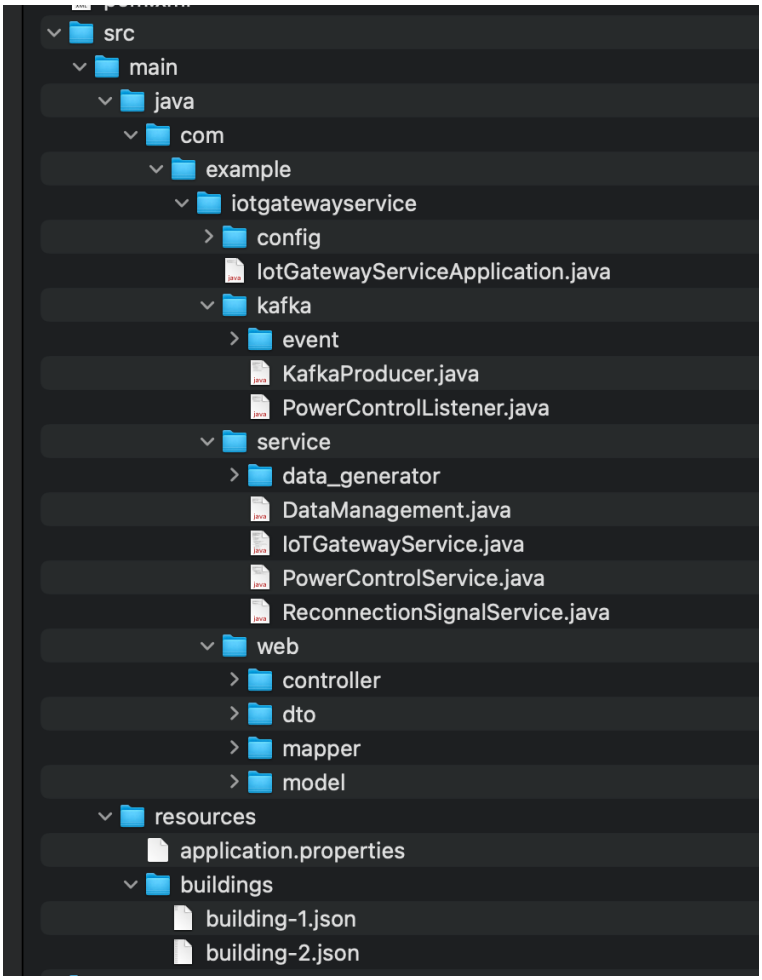

Рисунок 3.1 - Структура IoT Gateway Service

Структура сервісу IoT Gateway розподілена на декілька логічно відокремлених компонентів, серед яких – модуль генерації даних. Відсутність можливості збору реальних показників з датчиків вимагає імітації вхідних даних, для чого було розроблено спеціальний тестовий механізм. Цей механізм реалізований у вигляді компонентів, заснованих на класах PowerDataRandomizer.java та DataGenerationService.java, які відповідають за моделювання даних споживання енергії та формування відповідних подій у системі.

Окрім тестової генерації даних, сервіс IoT Gateway забезпечує читання та подальше використання конфігураційних файлів, що містять критично важливу інформацію про енергетичні параметри будинків і квартир. Дані конфігурації є спільними з сервісом Power Management і залучаються до роботи з моменту запуску сервісу IoT Gateway.

Важливим функціоналом DataGenerationService є ідентифікація ситуацій, коли генеровані дані вказують на можливе перевантаження резервного джерела енергії. У разі виявлення такої ситуації сервіс припиняє генерацію даних і ініціює подію «OVERLOADED», що сигналізує про аварійне відключення генератора. Механізм PowerDataRandomizer виконує важливу функцію у формуванні реалістичного потоку даних, використовуючи попередні дані споживання для генерації послідовних значень, які відтворюють реальні шаблони споживання енергії, як показано на рисунку 3.2.

```
1 usage   dmuntian *
public class PowerDataRandomizer {
    1 usage
    private static final double MAX_CONSUMPTION_CHANGE = 3000.0;
    1 usage
    private static final double CHANGE_PROBABILITY = 0.6;

    1 usage   dmuntian *
    public static double nextDouble(Double lastValue, double maxConsumption, double growProbability) {
        var random = ThreadLocalRandom.current();
        double newValue = lastValue != null ? lastValue : maxConsumption / 2;

        if (random.nextDouble() < CHANGE_PROBABILITY) {
            double change = (random.nextDouble() < growProbability ? 1 : -1) * random.nextDouble() *
                    MAX_CONSUMPTION_CHANGE;
            newValue += change;
            newValue = Math.max(15, Math.min(newValue, maxConsumption));
        }

        return Math.round(newValue * 100.0) / 100.0;
    }
}
```

Рисунок 3.2 - PowerDataRandomizer.java

Така система імітації дозволяє не лише перевірити роботу алгоритму в різноманітних сценаріях, але й забезпечує можливість глибокого аналізу реакції системи на критичні події, такі як відключення від енергомережі. Реалізація тестового генератора даних відіграє суттєву роль у валідації і верифікації функціональності розробленого програмного забезпечення.

Секція веб-інтерфейсу сервісу IoT Gateway (Рис. 3.3) виконує роль моста між реальними даними, які надходять від будинку, та алгоритмом обробки цих даних у системі. Веб-частина проекту репрезентує собою симуляційну заміну реальних даних, що передбачає наявність кількох REST-ендпоінтів:

- POST api/v1/signal: цей кінцевий пункт приймає сигнал від Smart Box користувача, що ініціює «готовність» домогосподарства до взаємодії з системою;
- POST api/v1/event: через цей ендпоінт система приймає різні події, такі як BLACKOUT, GENERATOR_ACTIVATED, OVERLOADED, або NORMALIZED;
- POST api/v1/data/stream: запускає процес генерації даних для конкретного дому, емулюючи підключення нового дому до системи у реальному світі.

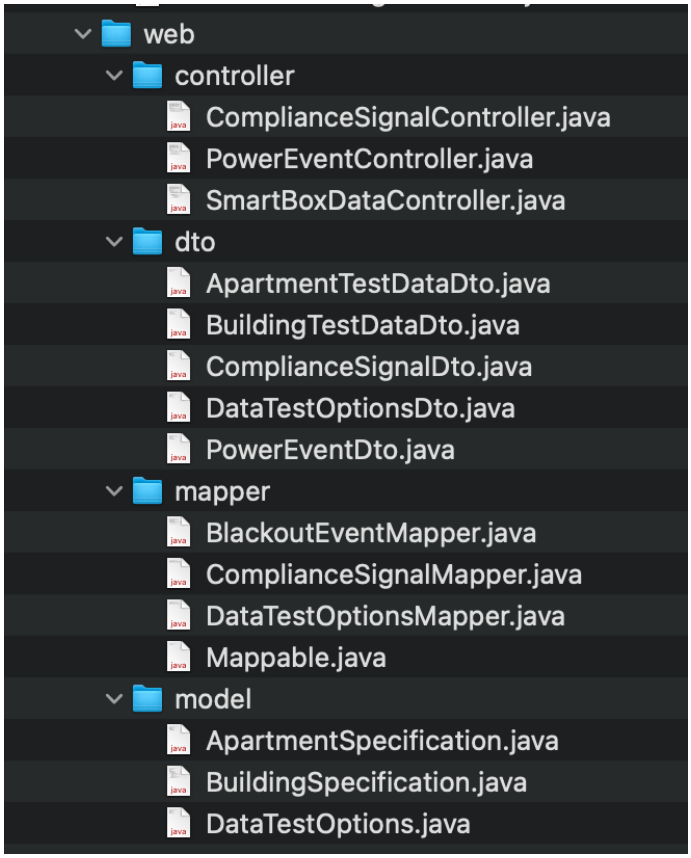

Рисунок 3.3 - Web інтерфейс IoT Gateway

Далі, проект включає інтеграцію з системою Apache Kafka (Рис. 3.4) [19], де описані моделі подій, які використовуються для взаємодії з Kafka. Клас KafkaProducer відповідає за відправлення повідомлень до Kafka, тоді як клас PowerControlListener слухає топік "power-control", куди сервіс Power Management відправляє повідомлення для управління підключенням квартир до мережі.

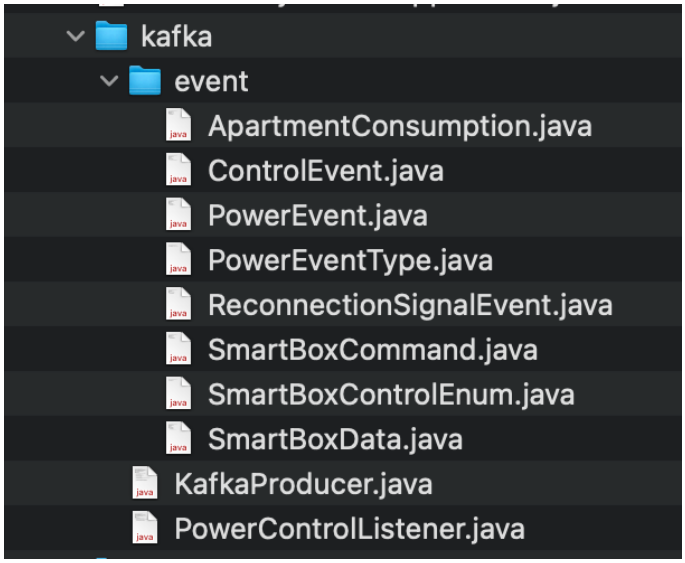

Рисунок 3.4 - Директорія Kafka

Важливою складовою є директорія сервісів (Рис. 3.5), де розміщена вся бізнес-логіка сервісу IoT Gateway. Центральне місце займає основний сервіс IoTGatewayService, що реалізує логіку потокової передачі даних та емуляцію реального споживання енергії будинком. Він також обробляє PowerEvent'и, реагуючи на події, як-от OVERLOADED, тим самим припиняючи генерацію споживаної потужності для квартир, імітуючи реальні реакції системи на екстрені події.

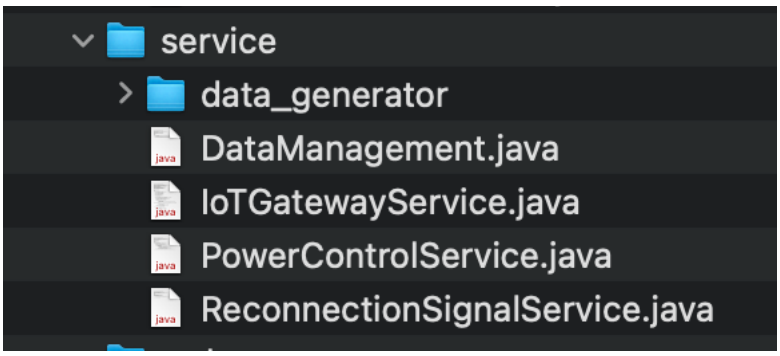

Рисунок 3.5 - Сервіси IoT Gateway

Завершується огляд налаштуваннями сервісу (Рис. 3.6), представленими у файлі application.properties, де зазначені усі конфігураційні параметри проекту, що є вирішальними для адаптації системи до специфічних умов експлуатації.

```
server.port=8081
kafka.topic.iot.data.name=power-data
kafka.topic.iot.data.frequency=500
kafka.topic.iot.power-event.name=power-event
kafka.topic.iot.power-control.name=power-control
kafka.topic.iot.button-signal.name=button-signal
spring.kafka.bootstrap-servers=localhost:9093
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.producer.acks=1
spring.kafka.producer.properties.spring.json.add.type.headers=false
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.group-id=iot-gateway-consumer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.consumer.auto-offset-reset=earliest
logging.level.org.apache.kafka=WARN
logging.level.org.springframework.kafka=WARN
logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS
logging.pattern.level=%5p
logging.pattern.console=%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} | %clr(%5p) | %clr(%class{0}){cyan} | %m%n
```

Рисунок 3.6 - application.properties

3.2 Power Management сервіс

У серці реалізації інтелектуальної системи керування споживанням електроенергії виступає сервіс Power Management, який виконує функції центрального вузла обробки даних та управління. Структура проекту management-service відображає ієрархію та зв’язки між різними компонентами, що забезпечують зв'язок з системою Apache Kafka для обміну подіями і сигналами (Рис. 3.7)

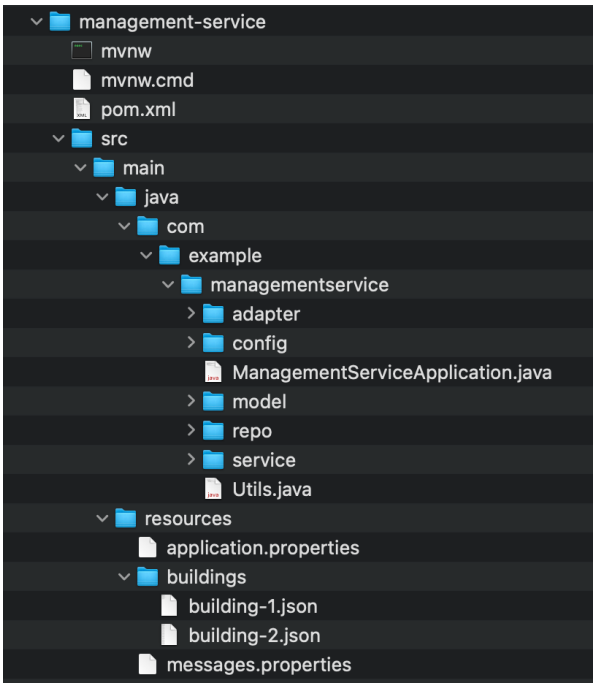

Рисунок 3.7 - Структура PowerManagementService

Сегмент adapter (Рис. 3.8) інкапсулює моделі даних та відповідає за інтеграцію з Kafka, отримуючи та надсилаючи івенти. Він включає в себе IoT-моделі, які визначають структуру даних, отриманих від IoT Gateway, control івенти, що керують процесами підключення та відключення квартир до електромережі, а також notification івенти, які служать для комунікації з мешканцями. Ключовий компонент PowerEvent дозволяє системі реагувати на зміни стану електромережі, як то виникнення блекауту чи активація генератора, а ReconnectionSignalEvent це сигнали від мешканців про готовність до підключення..

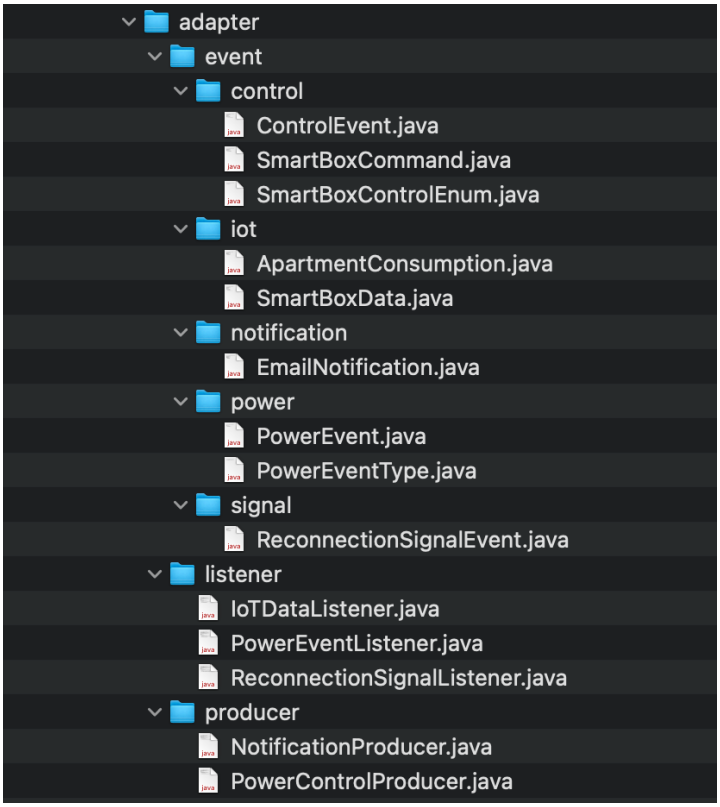

Рисунок 3.8 - adapters

Роль слухачів та продюсерів даних до Kafka у цій частині структури немаловажна, оскільки вони забезпечують здатність системи реагувати на потокові дані та оперативно передавати команди на виконання. Загалом, описана структура Power Management Service не тільки відображає внутрішню організацію сервісу, але й ілюструє його функціональну взаємодію з іншими елементами системи. Топіки, що слухає сервіс: power-data, power-event, button-signal. Топіки, в які пише сервіс: power-control та notification.

Модель даних Smart Box, що сервіс отримує в потоковому режимі виглядає так як показано на рисунку 3.9.

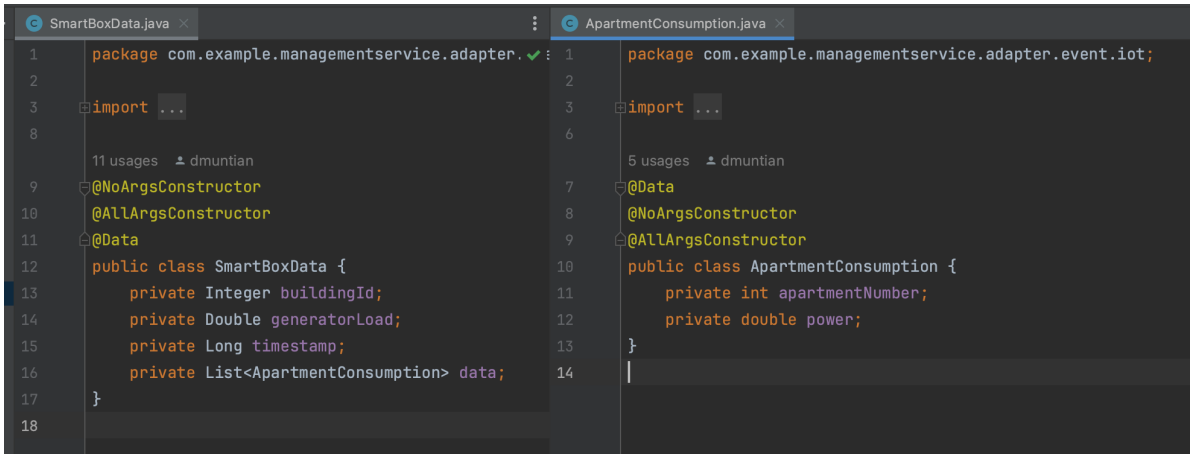

Рисунок 3.9 - IoT дані Smart Box

Далі йде директорія model (Рис. 3.10)

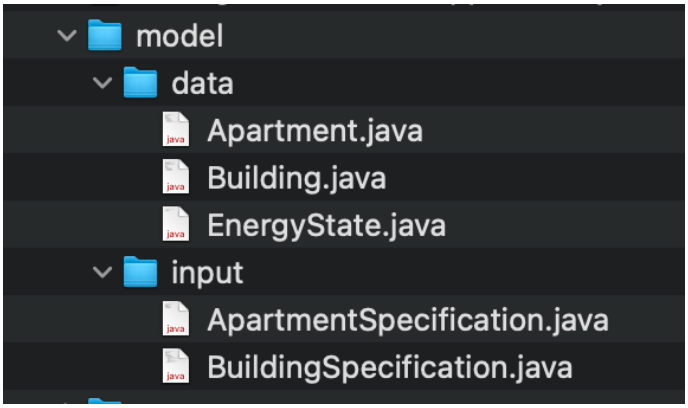

Рисунок 3.10 - Директорія model

В цій директорії описані моделі даних, що зберігаються в Redis, та якими оперує система на протязі всієї роботи алгоритму. Кожен клас відповідає за своє специфічне завдання в системі. Apartment уособлює собою дані про квартиру та її поточний стан у контексті споживання енергії, включаючи обмеження під час відключення та інформацію про порушення лімітів споживання. Building інкапсулює інформацію про будівлю в цілому, включаючи її стан відносно енергопостачання і зберігає дані про всі апартаменти, що входять до складу будинку. EnergyState відображає стан енергопостачання будівлі. А ApartmentSpecification та BuildingSpecification визначають параметри для ініціалізації вищезгаданих об'єктів. На рисунку 3.11 проілюстровано, як ці дані зберігаються в базі даних Redis.

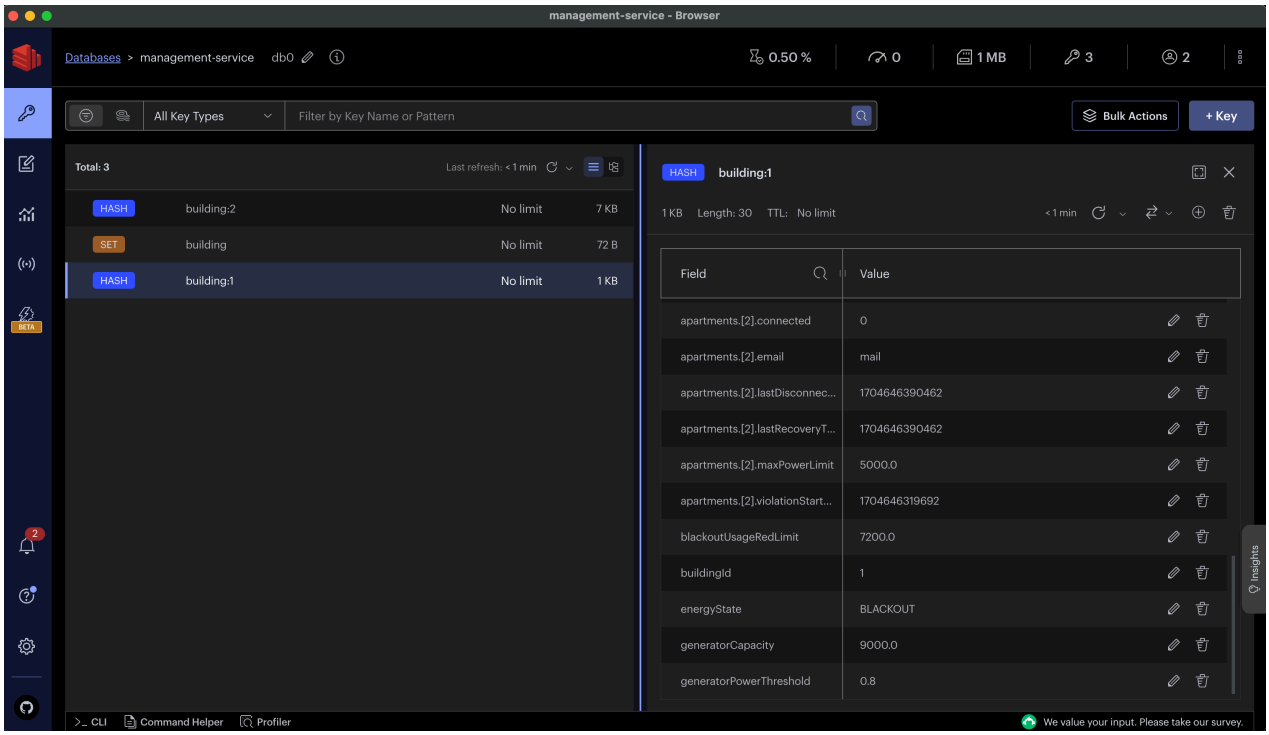

Рисунок 3.11 - Redis модель даних

Архітектура Power Management Service передбачає систематичний прийом і обробку даних від Smart Box. Структурний розділ model містить схеми даних, які функціонують як фундаментальна основа для всієї системи, забезпечуючи зберігання важливої інформації у базі даних Redis. Зокрема, у цій частині знаходяться моделі, що репрезентують структуровані дані про енергетичні показники будинку, генератора, окремих квартир, що потрібні для аналізу та відповідних дій у разі непередбачених змін в електромережі.

Основною частиною сервісу традиційно виступають Spring сервіс (Рис. 3.12).

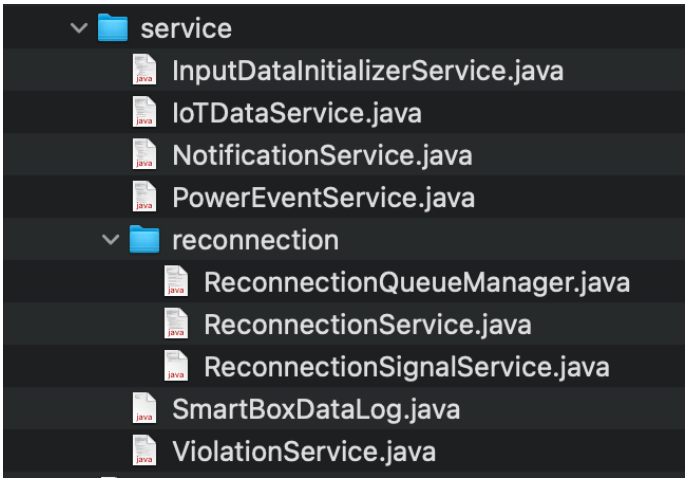

Рисунок 3.12 - Spring сервіси

Ключові компоненти, такі як ViolationService та ReconnectionService, виконують стратегічно важливу роль у процесі моніторингу та реагування на відхилення в енергоспоживанні квартир. Вони ініціюють процедури перевірки дотримання енергетичних норм та керують процесом підключення квартир до електромережі відповідно до пріоритету, зазначеного в алгоритмі роботи системи.

Огляд традиційно закінчується конфігурацією сервісу (Рис. 3.13):

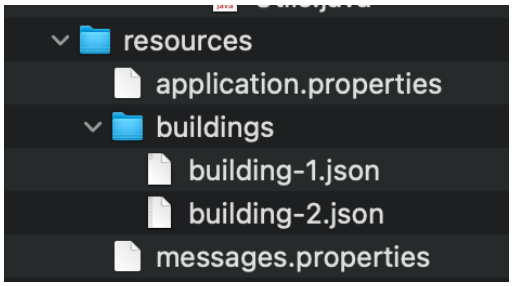

Рисунок 3.13 - Структура конфігурації сервісу

Файли конфігурації, що розташовані в директорії buildings, включають в себе детальну інформацію про енергетичні параметри будинків, які використовуються для ініціалізації та налаштування сервісу (Рис.3.14).

Рисунок 3.14 - Building-1.json

Файл application.properties містить налаштування, необхідні для коректної роботи Spring фреймворку (Рис. 3.15).

```
server.port=8082
kafka.topic.iot.data.name=power-data
kafka.topic.iot.power-event.name=power-event
kafka.topic.iot.power-control.name=power-control
kafka.topic.iot.button-signal.name=button-signal
kafka.topic.iot.notification.name=notification
spring.kafka.bootstrap-servers=localhost:9093
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.group-id=iot-consumer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.springframework.kafka.support.serializer.JsonSerializer
spring.kafka.producer.acks=1
spring.kafka.producer.properties.spring.json.add.type.headers=false
logging.level.org.apache.kafka=WARN
logging.level.org.springframework.kafka=WARN
logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS
logging.pattern.level=%5p
logging.pattern.console=%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} | %clr(%5p) | %clr(%class{0}){cyan} | %m%n
spring.data.redis.host=localhost
spring.data.redis.port=6389
management.violation.max-period=20000
management.connection.attempt.min-interval=40000
```

Рисунок 3.15 - Конфігурація Spring проекту

Messages.properties забезпечує зберігання текстів повідомлень, які будуть використовуватися для нотифікації споживачів (Рис. 3.16).

```
notification.message.reconnecting.text=Apartment {0}-{1} has been reconnected to the power network.\n\
  Please keep your consumption within the limit (up to {2} watts).
notification.message.reconnecting.subject=Apartment {0}-{1} - reconnection

notification.message.restored.text=Electricity has returned, and we have successfully transitioned to the main power source
notification.message.restored.subject=Apartment {0}-{1} - power restored

notification.message.overloaded.text=Attention! The backup power source was emergency shut down due to excessive consumption.\n\
  The generator will be restarted shortly.\n\
  To prevent this from happening again, please adhere to the consumption limits!
notification.message.overloaded.subject=Apartment {0}-{1} - overload
```

Рисунок 3.16 - Приклад повідомлень

Отже, даний розділ представляє собою деталізований опис ключових аспектів імплементації Power Management Service, який виступає в ролі центрального процесора для обробки та керування потоками даних у системі інтелектуального керування споживанням електроенергії.

3.3 Notification сервіс

Сервіс сповіщень, що називається Notification Service, виконує вузько спеціалізовану функцію в рамках загальної системи управління

електроспоживанням. Його основна роль полягає у прийманні комунікаційних звернень із топіка notification системи Kafka та відповідному розсиланні електронних листів користувачам (Рис. 3.17).

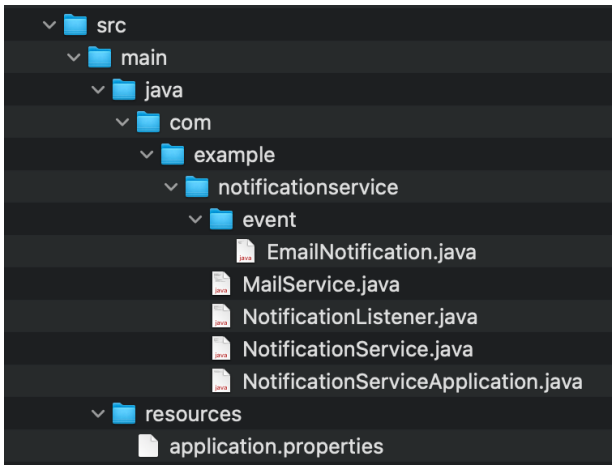

Рисунок 3.17 - Структура Notification Service

Структура сервісу сповіщень є оптимізованою для виконання задачі розсилання електронних листів, використовуючи модуль spring-boot-starter-mail. Цей модуль інтегровано в проект для ефективної взаємодії з електронною поштою, надаючи можливість MailService ініціювати та відправляти повідомлення (Рис. 3.18).

```java
@Service
@Slf4j
@RequiredArgsConstructor
public class MailService {
    private final JavaMailSender javaMailSender;

    @Value("digitalcamera.factory@gmail.com")
    private String mailFrom;

    1 usage  dmuntian
    public void sendMail(EmailNotification notification) {
        try {
            var mimeMessage = javaMailSender.createMimeMessage();
            var helper = new MimeMessageHelper(mimeMessage, "utf-8");
            helper.setFrom(mailFrom, "Power Consumption Management Service");
            helper.setTo(notification.getEmail());
            helper.setSubject(notification.getSubject());
            var text = notification.getText().replace("\n", "<br>");
            helper.setText(text, true);
            javaMailSender.send(mimeMessage);
        } catch (MailSendException | MessagingException | UnsupportedEncodingException e) {
            log.error("Failed to send email to {}: {}", notification.getEmail(), e.getMessage());
        }
    }
}
```

Рисунок 3.18 - Відправка повідомлень

Файл конфігурації сервісу (Рис. 3.19) встановлює параметри з'єднання та аутентифікації, що необхідні для взаємодії з поштовим сервером. Це дозволяє сервісу ефективно обробляти вихідні електронні листи, гарантуючи, що вони будуть доставлені до одержувачів у відповідь на реакції алгоритму системи.

```
server.port=8083
kafka.topic.iot.notification.name=notification
spring.kafka.bootstrap-servers=localhost:9093
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.springframework.kafka.support.serializer.JsonDeserializer
spring.kafka.consumer.group-id=iot-notification-consumer
spring.kafka.consumer.properties.spring.json.trusted.packages=*
spring.kafka.consumer.auto-offset-reset=earliest
logging.pattern.dateformat=yyyy-MM-dd HH:mm:ss.SSS
logging.pattern.level=%5p
logging.pattern.console=%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} | %clr(%5p) | %clr(%class{0}){cyan} | %m%n

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=digitalcamera.factory@gmail.com
spring.mail.password=[redacted]
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

Рисунок 3.19 - Конфігурація Notification сервісу

Таким чином, Notification Service служить як критично важливий інтерфейс між системою управління електроенергією та кінцевими користувачами, забезпечуючи швидкий та надійний зв'язок через електронну пошту.

3.4 Конфігурація зовнішніх залежностей

Використання Docker-compose (Рис. 3.20) є ключовим елементом у спрощенні процесу розгортання та управління мультиконтейнерними додатками, оскільки воно дозволяє описувати і запускати складні програмні середовища в ізольованих контейнерах за допомогою простого файлу YAML.

docker-compose.yml

```yaml
version: '2'
services:
  zookeeper:
    image: 'wurstmeister/zookeeper:latest'
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: 'wurstmeister/kafka:latest'
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "9093:9093"
      - "9997:9997"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT
      KAFKA_LISTENERS: "INTERNAL://:9092,EXTERNAL://:9093"
      KAFKA_ADVERTISED_LISTENERS: INTERNAL://kafka:9092,EXTERNAL://localhost:9093
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_JMX_PORT: 9997
      KAFKA_JMX_HOSTNAME: kafka
      KAFKA_CREATE_TOPICS: "power-data:5:1,power-event:1:1,button-signal:5:1,power-control:5:1,notification:5:1"

  redis:
    image: "redis:latest"
    hostname: redis
    container_name: redis
    ports:
      - "6389:6379"

  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - 8090:8080
    environment:
      DYNAMIC_CONFIG_ENABLED: true
```

Рисунок 3.20 - Конфігурація docker-compose

Файл docker-compose.yml включає в себе визначення всіх служб, що становлять архітектуру системи, включаючи бази даних, брокери повідомлень та інші компоненти. Кожен компонент у файлі docker-compose.yml має вказані версії образів Docker, параметри зберігання (volumes), мережеві налаштування (networks) та змінні середовища (environment variables), які забезпечують необхідну конфігурацію для їхньої взаємодії та коректної роботи.

Кожний компонент конфігурації служби у файлі docker-compose.yml відіграє свою роль:

- Kafka: Як центральний компонент для обробки потокових даних, Kafka вимагає налаштування змінних середовища для координації роботи в кластері, забезпечення безпеки з'єднань та управління розділами топіків.
- Zookeeper: Цей сервіс використовується для управління станом в кластері Kafka, зокрема для координації брокерів та ведення реєстру номінацій.
- База даних (Redis): Використовується для зберігання стану системи та іншої важливої інформації, що потрібна для функціонування сервісів.

Змінні середовища для Kafka можуть включати такі параметри, як конфігурація реплікації, час зберігання повідомлень, ліміти розміру топіків та інші налаштування, пов'язані з продуктивністю та відмовостійкістю.

Використання Docker-compose у цьому контексті не лише забезпечує ефективне управління зовнішніми ресурсами, але й спрощує процес розгортання, оскільки всі сервіси можуть бути запущені однією командою. Це важливо для забезпечення консистентності середовища в різних етапах розробки — від розробника до продакшну. Також це дає можливість швидкого масштабування системи, додаючи або видаляючи контейнери з сервісами в залежності від потреб.

3.5 Приклад роботи алгоритму

В даном розділі приводиться приклад роботи системи. В якості демонстрації роботи будуть використані записи з логів Power Management сервісу.

Отже, початок роботи. Конфігурація будинку, що тестується виглядає наступним чином:


```
{
 "buildingId": 1,
 "generatorCapacity": 9000,
 "generatorPowerThreshold": 0.8,
 "maxCommunalPowerUsage": 200,
 "apartments": [
   {
    "apartmentNumber": 1,
    "maxPowerLimit": 3500,
    "email": "daniilmuntjan@gmail.com"
   },
   {
    "apartmentNumber": 2,
    "maxPowerLimit": 4000,
    "email": "mail"
   },
   {
    "apartmentNumber": 3,
    "maxPowerLimit": 5000,
    "email": "mail"
   }
 ]
}
```


Будинок підключено до системи, починається потокова передача даних (так як в нас тестове середовище - надіслано POST запит на IoT Gateway api/v1/data/stream з айді будинком – 1). Отримуємо такі івенти раз на секунду:


```
{
 "generator" : 0.0,
 "apartments" : [ 3500.0, 3868.0, 4986.19 ],
 "timestamp" : "2024-01-04 16:45:58.455"
}
```

Відбувається відключення від міської електромережі (надсилаю POST запит на api/v1/event IoT Gateway сервісу з івентом BLACKOUT). Реакція сервісу Power Management можна побачити на рисунку 3.21.

```
WARN | PowerEventService | BLACKOUT detected (building 1)
WARN | NotificationService | [Notify building 1] - blackout
INFO | ViolationService | Building 1 - disconnecting violators
WARN | NotificationService | [Notify apartment 1-1] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-1 is added to no-response queue
WARN | NotificationService | [Notify apartment 1-2] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-2 is added to no-response queue
WARN | NotificationService | [Notify apartment 1-3] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-3 is added to no-response queue
```

Рисунок 3.21 - Реакція алгоритму на блекаут

Тобто, як було зазначено в алгоритмі, програма проаналізувала споживання на момент безпосередньо перед блекаутом, та відключила всіх умовних порушників від мережі, в даному випадку ними виявилися всі 3 квартири. Як видно з конфігурації, для першої квартири була проставлена моя особиста пошта як спосіб комунікації, на яку прийшли два сповіщення (Рис. 3.22), що відповідає логам програми вище.

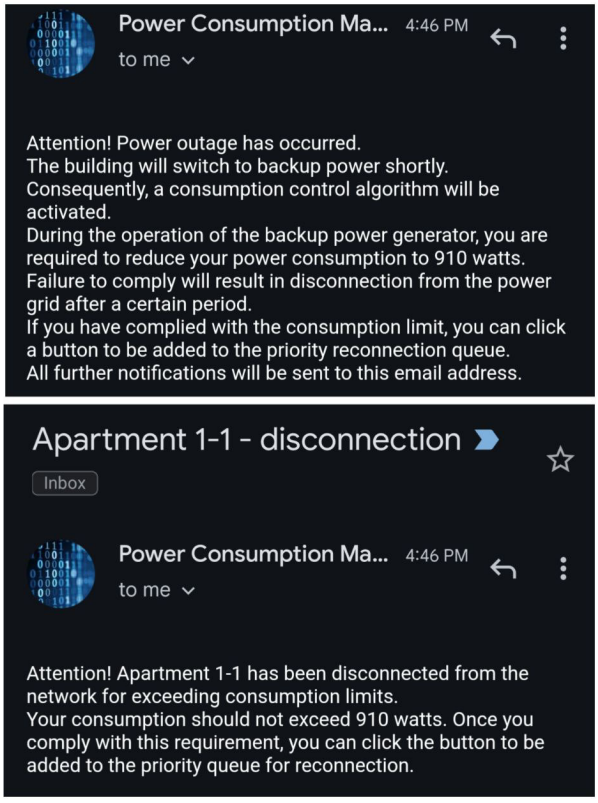

Рисунок 3.22 - Сповіщення про відключення

Далі, шляхом тестового надсилання POST запиту api/v1/event із значенням GENERATOR_ACTIVATED, симулюється робота АВР та запуск резервного джерела живлення – генератора. Головний сервіс системи відреагував: «WARN | PowerEventService | GENERATOR ACTIVATED (building 1)».

Нагадаю деякі конфігураційні змінні – це management.validation.max-period, що дорівнює 20 секундам (для тесту), це час, який квартира може вважатися порушником та не бути відключеною від мережі (після його завершення – вона відключається насильно), та management.connection.attempt.min-interval, що дорівнює 40 секунд – це мінімальний час, який квартира-порушник повинна бути відключеною, після якого алгоритм може розглянути її як кандидата на підключення в рамках черги.

Таким чином, після того, як завівся генератор, основний сервіс почав отримувати наступні івенти:


```
{
 "generator" : 194.6,
 "apartments" : [ 0.0, 0.0, 0.0 ],
 "timestamp" : "2024-01-11 16:46:07.938"
}
```


Тут навантаження генератора йде суто на загальнодомові потреби, адже всі три квартири поки що відключені від мережі.

З кожним отриманням Smart Box даних, алгоритм бере першу в черзі квартиру (в нашому випадку це квартира 1), та перевіряє, чи пройшло достатньо часу, щоб розглянути її як кандидата на підключення.

В цей час, мешканець квартири 2 натискає на кнопку «готовності» (Рис. 3.23).

```
INFO | ReconnectionService | Apartment 1-1 can be considered for reconnection in 25 seconds
INFO | ReconnectionQueueManager | Apartment 1-2 is added to priority queue
```

Рисунок 3.23 - Мешканець квартири 2 натиснув радіо кнопку

Як бачимо, він був перенесений до пріоритетної черги, відтепер першою в черзі є не квартира 1, а квартира 2: «Apartment 1-2 can be considered for reconnection in 24 seconds».

Після того, як проходить достатньо часу, починаються підключення квартир з черги (Рис. 3.24)

```
 WARN | NotificationService | [Notify apartment 1-2] - connecting
 INFO | ReconnectionQueueManager | Apartment 1-2 was removed from queue
 INFO | IoTDataService | {

i1.64, 0.0 ],
.1 16:46:39.936"

 WARN | NotificationService | [Notify apartment 1-1] - connecting
 INFO | ReconnectionQueueManager | Apartment 1-1 was removed from queue
 INFO | IoTDataService | {

, 151.64, 0.0 ],
.1 16:46:40.939"

 WARN | NotificationService | [Notify apartment 1-1] - you have become power violator! 20 seconds to decrease consumption!
 WARN | NotificationService | [Notify apartment 1-3] - connecting
 INFO | ReconnectionQueueManager | Apartment 1-3 was removed from queue
```

Рисунок 3.24 - Відбулось підключення квартир з черги

Як бачимо, в той же час квартира 1 стала «порушником», та їй було надіслано відповідне повідомлення (Рис. 3.25).

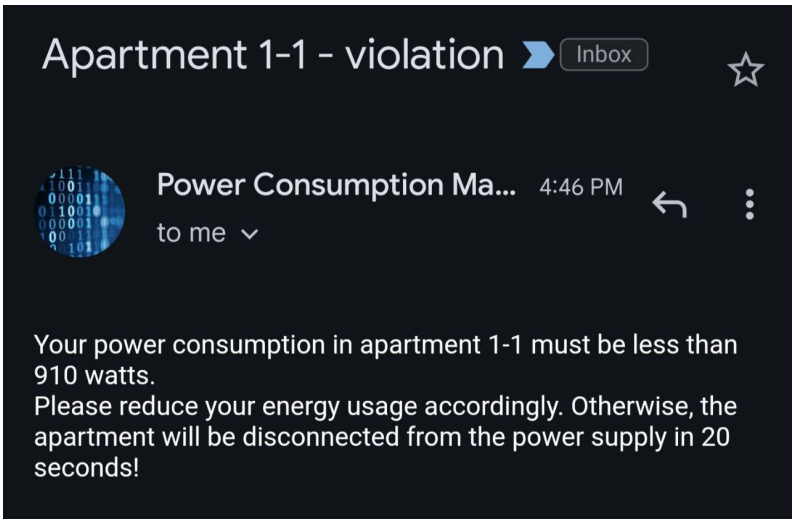

Рисунок 3.25 - Повідомлення про порушення

Через деякий час, всі три квартири почали порушувати обмеження споживання настільки різко та стрімко, що було досягнуто номінальне споживання генератору, та відбулось його аварійне відключення (рисунок 3.26).

```
WARN | ViolationService | Apartment 1-1 is still violating (duration: 1 second)
WARN | NotificationService | [Notify apartment 1-2] - you have become power violator! 20 seconds to decrease consumption!
WARN | NotificationService | [Notify apartment 1-3] - you have become power violator! 20 seconds to decrease consumption!
WARN | PowerEventService | GENERATOR OVERLOADED (building 1)
WARN | NotificationService | [Notify building 1] - generator overloaded
INFO | ReconnectionQueueManager | Queue was cleared for building 1
WARN | NotificationService | [Notify building 1] - blackout
INFO | ViolationService | Building 1 - disconnecting violators
WARN | NotificationService | [Notify apartment 1-1] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-1 is added to no-response queue
WARN | NotificationService | [Notify apartment 1-2] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-2 is added to no-response queue
WARN | NotificationService | [Notify apartment 1-3] - disconnecting for violation!
INFO | ReconnectionQueueManager | Apartment 1-3 is added to no-response queue
```

Рисунок 3.26 - Аварійне відключення генератора

Як бачимо, програма отримала цей сигнал від IoT Gateway та почала весь процес заново («Queue was cleared for building 1»), знову проаналізувавши порушників на момент відключення генератора, та превентивно їх відключивши, щоб після роботи АВР – генератор запустився. Також всім квартирам було розіслане сповіщення про перевантаження генератора (Рис. 3.27)

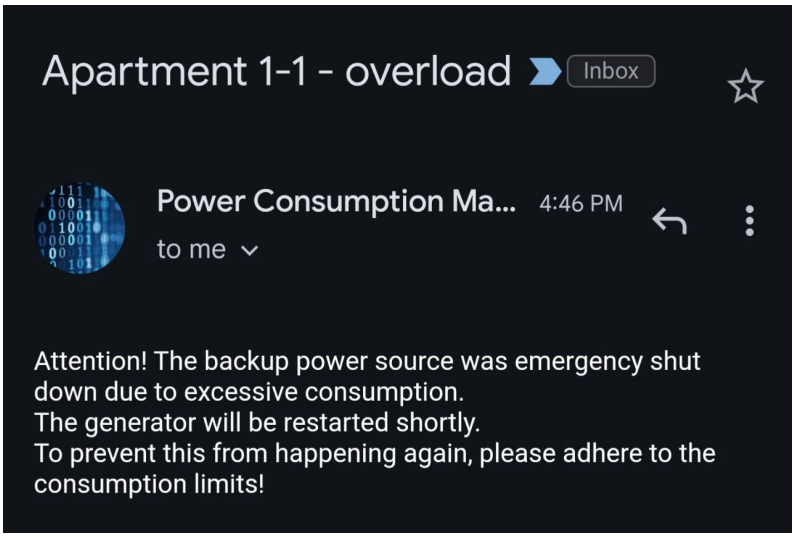

Рисунок 3.27 - Сповіщення про перевантаження генератора

Після активації генератора (надсилання тестового POST GENERATOR_ACTIVATED запиту на api/v1/event до IoT Gateway), система продовжила роботу як це було спочатку блекауту:

```
{
  "generator" : 127.23,
  "apartments" : [ 0.0, 0.0, 0.0 ],
  "timestamp" : "2024-01-11 16:47:27.314"
}
```

Через деякий час міська електромережа знову запрацювала, відпрацювала система АВР, та програма отримала про це сповіщення (Рис. 3.28)

```
WARN | PowerEventService | POWER RESTORED (building 1)
WARN | NotificationService | [Notify building 1] - power restored
INFO | ReconnectionQueueManager | Queue was cleared for building 1
INFO | PowerEventService | BUILDING STATE: NORMALIZED
```

Рисунок 3.28 - Мережа відновилась

Відповідно маємо розсилку email, що проілюстровано на рисунку 3.29.

Рисунок 3.29 - Сповіщення про відновлення електропостачання

Як видно з логів програми, всі стани були оновлені, квартири підключені IoT інформація більше не обробляється алгоритмом. Тепер система отримує дані від датчиків та ігнорує їх:

{
"generator" : 0.0,
"apartments" : [1025.59, 1777.41, 15.0],
"timestamp" : "2024-01-11 16:47:33.819"
}

3.6 Висновок

Отже, демонстрація роботи системи інтелектуального контролю споживання електроенергії у багатоквартирному будинку на основі аналізу логів Power Management сервісу підтвердила ефективність розробленого алгоритму. Сценарій реакції системи на випадок відключення міської електромережі та

подальше перемикання на резервне джерело живлення (генератор) виявився адекватним та відповідним поставленим вимогам.

Першочергове відключення квартир-порушників, які перевищували дозволений ліміт споживання, забезпечило зниження навантаження на генератор та його стабільну роботу. Процес відновлення живлення квартир, виходячи з черговості та пріоритетності, демонструє гнучкість та адаптивність системи до змінних умов. Особливо важливим є впровадження механізму «радіокнопки», який дозволяє мешканцям підтвердити готовність до зниження споживання, що сприяє ефективному управлінню енергоресурсами в екстрених ситуаціях.

Аварійне відключення генератора через перевантаження та подальше відновлення роботи системи після нормалізації електропостачання свідчить про надійність та стійкість системи до критичних ситуацій. Це підкреслює значення вбудованих механізмів моніторингу та відповіді на зміни в умовах експлуатації.

На основі аналізу результатів реалізації системи можна з впевненістю стверджувати, що всі вимоги до системи, зокрема ті, що описані в другому розділі дисертації, були виконані. Визначені теоретичні принципи роботи, її архітектурні особливості та програмні механізми були успішно інтегровані та протестовані у реальних умовах, що дозволило не тільки підтвердити їх ефективність, але й продемонструвати високу адаптивність системи до змінних обставин. Ключові аспекти, такі як алгоритм реакції на відключення електроенергії, управління навантаженням генератора, та механізми відновлення живлення, були реалізовані згідно з раніше визначеними специфікаціями.

Враховуючи важливість надійності та стабільності енергопостачання для житлових будинків, впровадження розробленої системи має потенціал стати значущим внеском у підвищення якості життя громадян та забезпечення енергетичної безпеки в Україні. Таким чином, результати тестування підкреслюють необхідність подальшого розвитку системи, її удосконалення та адаптації.

4 РОЗРОБКА СТАРТАП ПРОЕКТУ

4.1 Опис ідеї стартапу

Основною ідеєю стартап-проекту є створення інноваційної системи контролю споживання електроенергії, яка дозволяє оптимізувати використання енергетичних ресурсів у багатоквартирних будинках. В основі проекту лежить застосування принципів інтелектуального аналізу даних та розподіленого управління для забезпечення ефективного та збалансованого розподілу електроенергії з урахуванням поточних потреб та можливостей кожної квартири.

Система передбачає використання набору смарт-пристроїв та IoT технологій для збору даних про споживання енергії в реальному часі, а також передбачає можливість адаптивної реакції на зміни в енергетичному ландшафті, такі як аварійні відключення. Центральна частина системи – алгоритмічно-розподілений сервіс управління, який аналізує отримані дані та приймає рішення щодо оптимізації споживання енергії, включаючи автоматичне регулювання навантаження та ефективне використання резервних джерел енергії.

Метою проекту є не лише зниження загального споживання електроенергії та покращення енергетичної ефективності, але й забезпечення вищого рівня комфорту для мешканців, допомагаючи їм усвідомлено ставитися до власного енергоспоживання. За допомогою інтерактивного сповіщень користувачі зможуть отримувати рекомендації щодо енергозбереження та вчасно реагувати на енергетичні події.

Реалізація даного проекту має великий потенціал у сфері управління енергетичними ресурсами, відкриваючи нові можливості для створення стійких та економічно ефективних рішень у сучасних житлових комплексах.

Розпочнемо опис ідеї стартап проекту на основі застосунку:

Таблиця 4.1.

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигода для користувача

Інтелектуальна система контролю споживання електроенергії в багатоквартирних будинках	Для мешканців багатоквартирних будинків	Забезпечення стабільного електропостачання під час відключень від основної мережі, уникнення перенавантаження генератора, підтримка оптимального рівня споживання енергії.
	Для керуючих компаній та ЖЕКів	Моніторинг та управління навантаженням на генератор під час аварійних ситуацій, підвищення надійності електропостачання, зменшення ризиків відмов у роботі критичних систем.
	Для розробників IoT та енергетичних систем	Розробка інноваційних рішень для керування електроенергією в аварійних умовах, впровадження розумних технологій для ефективного управління ресурсами будинку.

Наступним кроком в розробці стартап-проекту є визначення потенційних конкурентів на ринку та аналіз їхніх сильних та слабких сторін у порівнянні з розробленою системою. Проведений аналіз наведено в таблиці 4.2.

Таблиця 4.2

Порівняльний аналіз сильних, слабких та нейтральних характеристик ідеї проекту

Техноекономічні характеристики ідеї	(потенційні) товари/концепції конкурентів	Недоліки	Нейтральні сторони	Переваги
Масштабованість (extensibility) - висока	Розширення системи обмежено			+
Інтеграція з будь-яким	Інтеграція лише з Smart House			+

типом будинку				
Використання event-driven підходу	Альтернативний метод		+	
Вартість реалізації - дешева	Висока вартість реалізації та інтеграції з хмарними застосунками компаній			+
Інтерфейс моніторингу подій	Великі компанії мають свої додатки із зручним UI/UX	+		
Розподілена система (мікросервісний підхід)	Монолітний підхід			+

Отже, розроблена система вирізняється своєю високою масштабованістю та можливістю інтеграції з будь-яким типом будинку, незалежно від його початкової конфігурації. Використання event-driven підходу забезпечує високу швидкість реакції на зміни стану системи та ефективну обробку даних в реальному часі. Низька вартість реалізації робить систему доступною для широкого кола користувачів, включаючи малі житлові комплекси та приватні домоволодіння.

Інтерфейс моніторингу подій, хоча може здатися менш зручним у порівнянні з розробками великих компаній, все ж пропонує всі необхідні функції для ефективного управління ресурсами. Розподілена система, заснована на мікросервісному підході, забезпечує гнучкість управління, легкість масштабування та високу надійність в порівнянні з монолітними системами.

Загалом, розроблена система є конкурентоспроможною завдяки своїй здатності до швидкої адаптації під різноманітні умови використання та забезпечення стабільної роботи в аварійних ситуаціях, таких як блекаути. Вона відкриває нові можливості для управління споживанням електроенергії, роблячи цей процес більш ефективним та економічно вигідним для користувачів.

4.2 Технологічний аудит проекту

Вибір технологій для створення інтелектуальної системи контролю споживання електроенергії є ключовим фактором, який впливає на швидкість розробки, масштабування, а також на ефективність впровадження інновацій. Далі робимо аудит технологій проекту, який наведено у таблиці 4.3.

Таблиця 4.3.

Технології для проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Інтелектуальна система контролю споживання електроенергії	Spring Framework / Java	+	Не треба купляти, в доступі
2		Redis як база даних	+	Не треба купляти, в доступі
3		Apache Kafka	+	Не треба купляти, в доступі
4		Kafka UI для наочності роботи	+	Не треба купляти, в доступі
5		RedisInsight	+	Не треба купляти, в доступі

Підсумовуючи, обрані технології відкривають широкі можливості для створення інтелектуальної системи, здатної ефективно реагувати на виклики сучасного енергетичного господарства, забезпечуючи високу продуктивність, масштабованість, а також оперативність в екстрених ситуаціях.

4.3 Аналіз ринкових можливостей запуску стартап проекту

Дослідження ринку є ключовим аспектом для оцінки відповідності продукту потребам ринку, рівня конкуренції, а також для виявлення потенційних

можливостей. Нижче представлено аналіз ринку, що може бути використаний для оцінки ринкових перспектив розробленої інтелектуальної системи.

Таблиця 4.4.

Попередня характеристика потенційного ринку для стартап проекту:

№	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, $ млн	Понад 200
3	Динаміка ринку	Стабільне зростання на 10-20% на рік через посилення екологічних норм та стандартів
4	Наявність обмежень для входу	Помірні (необхідні інвестиції на розробку IoT частини, сертифікація)
5	Вимоги до стандартизації	Строгі вимоги до стандартів безпеки та ефективності систем
6	Середня рентабельність	Близько 30% залежно від масштабів впровадження та потимізації витрат

Основними факторами, що сприяють зростанню ринку є стрімке посилення вимог до енергоефективності будівель та пошук шляхів оптимізації споживання енергії. Високий рівень інтересу до інноваційних енергозберігаючих технологій та інтелектуального управління споживанням електроенергії в багатоквартирних будинках створює підґрунтя для успішного входу на ринок. Однак, високі обмеження при вході вимагають значних початкових інвестицій та ретельної підготовки продукту до впровадження.

В цілому, з огляду на постійно зростаючу потребу в оптимізації споживання електроенергії та економічної ефективності, проект інтелектуальної системи контролю споживання електроенергії має хороші перспективи на ринку. Наявність технологічних інновацій, висока масштабованість та адаптивність

системи до різних типів будівель та енергосистем, а також потенціал зниження експлуатаційних витрат роблять її привабливою для широкого кола споживачів. Незважаючи на строгі стандарти безпеки та ефективності, які вимагають додаткових інвестицій у дотримання нормативних вимог, середня рентабельність проекту є високою, що вказує на його високу конкурентоспроможність і перспективність.

Для визначення стратегії розробки та впровадження продукту необхідно зрозуміти основні потреби ринку та особливості цільових груп. Важливо визначити, які вимоги ставлять користувачі до системи, та як вони планують її використовувати. Інформація про потенційних користувачів зведена в таблицю 4.5.

Таблиця 4.5.

Характеристика потенційних клієнтів проекту

№	Потреба ринку	Цільова група	Особливості поведінки цільової групи	Вимоги споживачів до товару
1	Енергоефективність та надійність постачання енергії	Управляючі компанії багатоквартирних будинків	Шукають надійні рішення для економії енергії	Висока точність регулювання, інтеграція з іншими системами будинку
2	Автономність у критичних ситуаціях	Житлові комплекси, готелі	Потребують систем, що забезпечують автономне живлення при відключенні електрики	Надійність, швидка реакція на аварійні ситуації

	Моніторинг та оптимізація споживання	Офісні центри, великі торгові комплекси	Інтерес до зниження витрат на електроенергію	Інтеграція з системами "розумного будинку", зручність управління

З огляду на потреби в енергоефективності, автономності та зручності управління енергоспоживанням, інтелектуальна система контролю має високий потенціал затребуваності на ринку. Система повинна забезпечувати точне регулювання навантаження, інтеграцію з існуючими системами "розумного будинку", надійність у критичних ситуаціях та зручність управління для кінцевого користувача.

В цілому, продукт має високу цінність для різних груп споживачів, що ставлять перед собою завдання ефективного контролю та оптимізації споживання електроенергії, забезпечення автономності енергопостачання у випадку аварійних ситуацій, а також для зручного моніторингу енергоспоживання.

Далі, в таблиці 4.6 приведені деякі ймовірні фактори загроз та можливі реакції на них.

Таблиця 4.6.

Фактори загроз

№	Загроза	Зміст	Можлива реакція
1	Зміна законодавства	Внесення нових норм щодо споживання енергії може вимагати змін в алгоритмах системи	Слідкування за законодавчими змінами, гнучке оновлення алгоритмів

2	Технологічні зміни в інфраструктурі	Розвиток технологій "розумного будинку" може зробити деякі функції системи застарілими	Постійне вдосконалення системи, інтеграція з новітніми технологіями
3	Кіберзагрози	Атаки на мережеву інфраструктуру можуть порушити роботу системи	Розробка та впровадження заходів кібербезпеки, швидке реагування на інциденти
4	Конкуренція на ринку	Поява нових аналогічних систем може зменшити частку ринку	Аналіз ринку, пошук унікальних конкурентних переваг, маркетингові стратегії

Аналіз показує, що основні загрози для системи включають законодавчі зміни, технологічні нововведення, кіберзагрози, енергетичну нестабільність та ринкову конкуренцію. Ефективне управління цими ризиками через прогнозування, адаптацію та інновації може забезпечити стійкість та ефективність системи в довгостроковій перспективі.

Таблиця 4.7 можливостей для інтелектуальної системи контролю споживання електроенергії та стратегій реагування:

Таблиця 4.7.

Фактори можливостей

№	Можливість	Зміст	Можлива реакція
1	Зростаюча свідомість споживання енергії	Споживачі стають більш обізнаними щодо енергоефективності	Розробка функціоналу для надання аналітичної інформації споживачам
2	Підвищення вимог до енергозбереження	Влада вводить жорсткіші стандарти споживання енергії	Адаптація системи під нові стандарти, розширення алгоритмів оптимізації
3	Розвиток Інтернету речей (IoT)	Інтеграція IoT стає більш поширеною в будинках	Інтеграція з широким спектром IoT пристроїв, розширення можливостей системи
4	Державні субсидії	Урядові програми підтримки енергозбереження	Подання на отримання субсидій та грантів для розвитку

На сучасному ринку існують значні можливості для розвитку системи, які пов'язані з глобальними трендами енергозбереження, розвитком технологій та екологічних ініціатив. Важливим аспектом є здатність системи до швидкої адаптації під змінювані вимоги та інтеграції з новими технологічними рішеннями. Оптимізація споживання електроенергії та інтелектуальне

управління навантаженням стають все більш актуальними, що відкриває потенціал для зростання інноваційних проектів у цій сфері.

Таблиця 4.8.

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип: олігополія	Кілька великих гравців займають основну частку ринку	Розробка інноваційних рішень, які відрізняються від існуючих
Рівень: національний	Конкуренція в межах країни	Фокус на відповідність місцевим нормам та стандартам
Галузева: енергетика та автоматизація	Специфіка енергетичного сектору та автоматизованих систем	Поглиблена спеціалізація та адаптація під особливості енергетичної сфери
Видова: за технологіями	Конкуренція між різними технологічними підходами	Вибір передових та ефективних технологій
За перевагами: енергоефективність	Попит на рішення, що знижують споживання енергії	Оптимізація алгоритмів для максимальної енергоефективності
За інтенсивністю: висока	Активна конкуренція за ринкову частку	Активні маркетингові та продажні стратегії

Конкурентне середовище в області інтелектуальних систем контролю споживання електроенергії характеризується олігополією з високою ступенем спеціалізації, вимагаючи від компаній інноваційного підходу та високої енергоефективності продуктів. Споживачі в даній галузі цінують технологічні нововведення, що підвищують ефективність та надійність систем, а також

забезпечують відповідність місцевим стандартам. Основним завданням для нового гравця на ринку є розробка рішень, які будуть інноваційними, ефективними, та матимуть конкурентні переваги перед існуючими продуктами.

Таблиця 4.9

Стратегії впровадження стартапу на ринок інтелектуальних систем контролю споживання електроенергії

№	Стратегія	Ймовірність погодження інвесторів	Строк реалізації
1	Вихід з базовим набором компонентів	70%	4 місяці
2	Вихід з повним набором компонентів та додатковим функціоналом	30%	8-12 місяців
3	Поетапний вихід: спочатку база, потім додатковий функціонал	60%	6 місяців

Найбільш вигідною здається стратегія поетапного входу на ринок. Початок із базового набору функціоналу дозволить швидко привернути увагу перших користувачів і отримати зворотний зв'язок для подальшого розвитку. Це також зменшує ризики для інвесторів та дозволяє зберегти гнучкість у відповідь на потреби ринку. Поступове розширення функціоналу може бути здійснено після отримання перших відгуків і аналізу попиту, що дозволить краще адаптувати продукт до потреб цільової аудиторії.

4.4 Розробка ринкової стратегії проекту

Таблиця 4.10

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Управління багатоквартирних будинків та ЖКГ	Готові до інновацій	Високий	Висока	Складна
2	Розробники «розумних» будинкових систем	Потребують переконання	Помірний	Середня	Проста
3.	Приватні будинки з автономними системами живлення	Готові якомога швидше	Низький	Низька	Дуже складна

В якості цільових груп обрано групи 1 та 2, оскільки вони володіють високою готовністю до інновацій та мають помірний до високого попит на запропонований продукт. Група 3 потребує спеціального підходу та додаткових маркетингових зусиль.

Основні напрямки стратегії розвитку можуть включати:

- Фокус на надійності та ефективності системи в екстрених ситуаціях.

- Постійне оновлення та модернізація системи для адаптації до змінюваних потреб користувачів та технологій.
- Тісна співпраця з управліннями будинків та ЖКГ для інтеграції системи та покращення її функціональності.

Стратегія конкуренції може включати:

- Активну підтримку та консультації для користувачів системи.
- Гнучкість та масштабованість рішень для задоволення потреб різних типів будівель.
- Підкреслення унікальних переваг системи, таких як можливість швидкого реагування на аварійні ситуації та автоматизація процесів управління споживанням енергії.

4.5 Розробка маркетингової програми

Визначення особливостей системи та ключових пунктів, на яких буде зосереджена увага, є фундаментом для маркетингової стратегії та кампанії. Для розробленої системи контролю споживання електроенергії в багатоквартирних будинках, ключові особливості включають:

- Автоматизоване управління електроенергією для ефективного розподілу навантаження.
- Надійність та швидкість реагування на аварійні ситуації, зокрема при відключенні від міської мережі.
- Можливість масштабування системи для використання у різноманітних типах будівель.
- Інтеграція з сучасними IoT технологіями для забезпечення точного моніторингу та управління.

Система збуту для цього проекту має наступні характеристики:

- Специфіка закупівельної поведінки цільових клієнтів: Спрямованість на довгострокове співробітництво з компаніями управління багатоквартирними будинками та ЖКГ.

- Функції збуту, які має виконувати постачальник товару: Надання технічної підтримки, налаштування системи під конкретні потреби, регулярні оновлення та удосконалення.
- Глибина каналу збуту: Прямий продаж клієнтам без посередників.
- Оптимальна система збуту: Прямі переговори з управліннями будинків та ЖКГ, демонстрація системи на спеціалізованих виставках та форумах, використання цифрових каналів комунікації для залучення нових клієнтів.

Ці особливості та збутова стратегія дозволяють створити ефективну систему контролю споживання електроенергії, що є актуальною та конкурентоспроможною на ринку.

4.6 Висновок

Цей розділ присвячено аналізу можливостей впровадження розробленої інтелектуальної розподіленої системи контролю споживання електроенергії в багатоквартирних будинках як стартапу.

З проведеного аналізу стає зрозуміло, що існує потреба в ефективних рішеннях для контролю та оптимізації споживання електроенергії, особливо в ситуаціях аварійного відключення. Головною перевагою системи є її здатність швидко реагувати на екстрені ситуації, забезпечуючи стабільне енергопостачання, а також можливість масштабування під різноманітні типи житлових комплексів.

Основною цільовою групою для цього проекту є управління житловими комплексами та компанії, які займаються обслуговуванням багатоквартирних будинків. Вибір такої цільової аудиторії забезпечує наявність стабільного попиту та готовності до впровадження новітніх технологій.

Рекомендовано стратегію поетапного впровадження: на першому етапі необхідно запустити базовий функціонал системи для підтвердження її ефективності та надійності, а потім поступово розширювати можливості, адаптуючи її під різні умови експлуатації та потреби клієнтів.

Отже, враховуючи актуальність проблематики, наявність попиту на ринку та ретельно продуману стратегію впровадження, проект має усі шанси на успіх. Важливими факторами ефективного розвитку стануть якість та надійність системи, а також здатність швидко адаптуватися до змінних умов ринку та потреб користувачів.

5 ВИСНОВКИ

Проведене дослідження та розробка системи контролю споживання електроенергії в багатоквартирному будинку демонструє значний потенціал інтеграції сучасних технологій в області IoT та розподілених систем у повсякденне життя. Система, яка була сконструйована та реалізована в рамках цієї роботи, є втіленням передових практик мікросервісної архітектури та використання потужних інструментів обробки потокових даних.

Важливість цього проекту обумовлена в тому числі консультаціями з декількома інженерами-електриками та енергетиками, які підтвердили його практичну придатність та реальність впровадження. Дослідження показало, що реалізація такої системи не тільки можлива, а й необхідна для підвищення ефективності управління енергетичними ресурсами під час екстреної ситуації, зменшення витрат та покращення комфорту мешканців.

Використання мови програмування Java у поєднанні з Spring Framework дозволило створити стабільну та гнучку платформу розподіленої системи, а вибір Redis як основної бази даних забезпечив високу швидкість доступу до даних, необхідних для негайної реакції на зміни в споживанні енергії.

У проект закладено широкий діапазон можливостей для подальшого розвитку. Перспективи розширення системи включають в себе розгортання в хмарних сервісах, неперервну інтеграцію та доставку (CI/CD), розвиток користувацьких інтерфейсів (макетів) для кращого візуального представлення та тестування роботи системи. Також можливість інтеграції алгоритмів машинного навчання для прогнозування показників споживання відкриває нові горизонти для оптимізації енергетичного балансу та економії ресурсів.

Система демонструє велику кількість працездатних сервісів, кожен з яких відіграє важливу роль у загальній структурі. IoT Gateway, Power Management та Notification сервіси формують основу для ефективної взаємодії з кінцевим споживачем, забезпечуючи стабільне та контрольоване споживання при живленні від резервного джерела електроенергії.

Завдяки гнучкості та масштабованості розробленої системи, вона може бути адаптована до різних житлових комплексів та масштабів використання, що робить її важливим кроком у розвитку сучасних енергетичних систем. Подальше дослідження та розробка в даній області мають потенціал значно вплинути на енергетичний ландшафт, пропонуючи нові можливості для оптимізації та автоматизації управління енергоресурсами.

6 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schneider Electric – URL: https://dou.ua/lenta/articles/dou-projector-hurma/ (дата звернення: 18.11.2023).
2. Siemens Building Technology – URL: https://www.siemens.com/global/en/products/buildings.html (дата звернення: 18.11.2023).
3. What is HVAC – URL: https://www.trane.com/residential/en/resources/glossary/what-is-hvac/ (дата звернення 20.11.2023).
4. IoT Fundamentals – URL: https://nibmehub.com/opac-service/pdf/read/IoT%20Fundamentals.pdf (дата звернення: 01.12.2023).
5. Iotji – URL: https://iotji.io/ (дата звернення: 01.12.2023).
6. What is LoRaWAN – URL: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/ (дата звернення: 29.11.2023)
7. Види та особливості автоматичних вимикачів – URL: https://energokvant.com/yak-vibrati-avtomatichnij-vimikach (дата звернення: 04.12.2023)
8. Автоматичний ввід резерву – URL: https://uk.wikipedia.org/wiki/Автоматичний_ввід_резерву (дата звернення: 04.12.2023)
9. Принцип роботи АВР – URL: http://www.esludger.com.ua/uk/yak-vybraty-avr-vse-pro-avtomatychnyi-vvid-rezervu.html (дата звернення: 05.12.2023)
10. Turn Your Raspberry Pi Into a Server – URL: https://medium.com/geekculture/turn-your-raspberry-pi-into-a-server-to-run-your-java-spring-mvc-app-862214279587 (дата звернення: 05.12.2023)
11. Як працює дизельний генератор? – URL: https://rental-power.com.ua/ua/kak-rabotaet-dizelnyj-generator/ (дата звернення: 06.12.2023)

12. Functional and Nonfunctional Requirements – URL: https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/ (дата звернення: 06.12.2023).
13. What is service oriented architecture? – URL: https://aws.amazon.com/ru/what-is/service-oriented-architecture/ (дата звернення: 07.12.2023).
14. The benefits of Using SOA – URL: https://www.swisslog.com/en-us/case-studies-and-resources/blog/the-benefits-of-using-service-oriented-architecture-(soa) (дата звернення: 08.12.2023).
15. What are microservice? – URL: https://microservices.io/ (дата звернення: 06.12.2023).
16. Netflix Microservices – URL: https://netflixtechblog.com/tagged/microservices (дата звернення: 08.12.2023).
17. DOU. Починаємо роботу з Apache Kafka. Частина 1 – URL: https://dou.ua/forums/topic/39363/ (дата звернення: 07.12.2023).
18. Confluent Official Website – URL: https://www.confluent.io/ (дата звернення: 09.12.2023).
19. Data Processing with SMACK – URL: https://alibaba-cloud.medium.com/data-processing-with-smack-spark-mesos-akka-cassandra-and-kafka-c3bb55c73eb1 (дата звернення: 10.12.2023).
20. Why Does Java Remains So Popular – URL: https://blog.stackademic.com/why-does-java-remain-so-popular-f52ea7f3b264 (дата звернення: 10.12.2023).
21. Spring Official Website – URL: https://spring.io/ (дата звернення: 11.12.2023).
22. Redis Official Website – URL: https://redis.io/ (дата звернення: 15.12.2023).
23. Postgres Official Website – URL: https://www.postgresql.org/ (дата звернення: 15.12.2023).
24. MongoDB Official Website – URL: https://www.mongodb.com/ (дата звернення: 15.12.2023).